Bugzilla Documentation

Release 0.2

David Burns

October 08, 2015

Contents

1 Installing Bugsy **3**

2 Using Bugsy **5**
2.1 Getting a bug from Bugzilla 5
2.2 Creating a new bug 5
2.3 Searching Bugzilla 5
2.4 Comments 6

3 Indices and tables **619**

Python Module Index **621**

Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

CHAPTER 1

Installing Bugsy

To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

CHAPTER 2

Using Bugsy

2.1 Getting a bug from Bugzilla

Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

2.2 Creating a new bug

To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

2.3 Searching Bugzilla

To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

2.4 Comments

Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

2.4.1 Welcome to Bugsy!

Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy

To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla

Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug

To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla

To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments

Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

```
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
```

```
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")

bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Welcome to Bugsy! Bugsy is a tool that allows you to programmatically work with Bugzilla using its native REST API.

To use you will do

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug123456.status = 'RESOLVED'
bug123456.resolution = 'FIXED'
bugzilla.put(bug123456)
```

Installing Bugsy To install Bugsy, simply use pip or easy install

Pip

```
pip install bugsy
```

easy_install

```
easy_install bugsy
```

Using Bugsy

Getting a bug from Bugzilla Getting a bug is quite simple. Create a Bugsy object and then get the bug number that you want.

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
```

Creating a new bug To create a new bug, create a Bug object, populate it with the items that you need and then use the Bugsy object to put the bug into Bugzilla

```
import bugsy
bug = bugsy.Bug()
bug.summary = "I really realy love cheese"
bug.add_comment("and I really want sausages with it!")
```

```
bugzilla = bugsy.Bugsy("username", "password")
bugzilla.put(bug)
bug.id #returns the bug id from Bugzilla
```

Searching Bugzilla To search for bugs you will need to create a `Bugsy` object and then you can call *search_for* and chain the search. The `Search` API is a Fluent API - you just chain the items that you need and then call *search* when the search is complete.

```
import bugsy
bugzilla = bugsy.Bugsy()
bugs = bugzilla.search_for\
                .keywords("checkin-needed")\
                .include_fields("flags")\
                .search()
```

More details can be found in from the `Search` class

Comments Getting comments from a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
comments = bug.get_comments()
comments[0].text # Returns  "I <3 Sausages"
```

Adding comments to a bug

```
import bugsy
bugzilla = bugsy.Bugsy()
bug = bugzilla.get(123456)
bug.add_comment("And I love bacon too!")
```

To see further details look at:

Bugsy

class `bugsy.`**`Bugsy`**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

`__init__`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **`username`** – Username to login with. Defaults to None
- **`password`** – Password to login with. Defaults to None
- **`userid`** – User ID to login with. Defaults to None
- **`cookie`** – Cookie to login with. Defaults to None
- **`apikey`** – API key to use. Defaults to None.
- **`bugzilla_url`** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**`Bugsy`**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

`__init__`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **`username`** – Username to login with. Defaults to None
- **`password`** – Password to login with. Defaults to None
- **`userid`** – User ID to login with. Defaults to None
- **`cookie`** – Cookie to login with. Defaults to None
- **`apikey`** – API key to use. Defaults to None.
- **`bugzilla_url`** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`get`(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **`bug_number`** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

`put`(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **`bug`** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields (**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords (**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search ()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe (*start*, *end*)
When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**`Bugsy`**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

`__init__`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`get`(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.Search`(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__

list of weak references to the object (if defined)

assigned_to(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**`Bugsy`**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

`__init__`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **`username`** – Username to login with. Defaults to None
- **`password`** – Password to login with. Defaults to None
- **`userid`** – User ID to login with. Defaults to None
- **`cookie`** – Cookie to login with. Defaults to None
- **`apikey`** – API key to use. Defaults to None.
- **`bugzilla_url`** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`get`(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **`bug_number`** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

`put`(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **`bug`** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields (**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords (**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search ()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe (*start*, *end*)
When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**`Bugsy`**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

`__init__`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`get`(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.`**`Search`**(*bugsy*)

This allows searching for bugs in Bugzilla

`__init__`(*bugsy*)

Initialises the search object

Parameters **`bugsy`** – Bugsy instance to use to connect to Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`assigned_to`(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

`bug_number`(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **`bug_numbers`** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

`change_history_fields`(*fields*, *value=None*)

`include_fields`(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

`keywords`(**args*)

When search() is called it will search for the keywords passed in here

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

`search`()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**`Bugsy`**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

`__init__`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **`username`** – Username to login with. Defaults to None
- **`password`** – Password to login with. Defaults to None
- **`userid`** – User ID to login with. Defaults to None
- **`cookie`** – Cookie to login with. Defaults to None
- **`apikey`** – API key to use. Defaults to None.
- **`bugzilla_url`** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`get`(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **`bug_number`** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

`put`(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **`bug`** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields (**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords (**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search ()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe (*start*, *end*)
When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.`**`Search`**(*bugsy*)

This allows searching for bugs in Bugzilla

`__init__`(*bugsy*)

Initialises the search object

Parameters **`bugsy`** – Bugsy instance to use to connect to Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`assigned_to`(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

`bug_number`(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **`bug_numbers`** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

`change_history_fields`(*fields*, *value=None*)

`include_fields`(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

`keywords`(**args*)

When search() is called it will search for the keywords passed in here

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

`search`()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.Bugsy`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields (**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords (**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search ()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe (*start*, *end*)
When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**`Bugsy`**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

`__init__`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`get`(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.`**`Search`**(*bugsy*)

This allows searching for bugs in Bugzilla

`__init__`(*bugsy*)

Initialises the search object

Parameters **`bugsy`** – Bugsy instance to use to connect to Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`assigned_to`(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

`bug_number`(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **`bug_numbers`** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

`change_history_fields`(*fields*, *value=None*)

`include_fields`(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

`keywords`(**args*)

When search() is called it will search for the keywords passed in here

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

`search`()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**`Bugsy`**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

`__init__`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **`username`** – Username to login with. Defaults to None
- **`password`** – Password to login with. Defaults to None
- **`userid`** – User ID to login with. Defaults to None
- **`cookie`** – Cookie to login with. Defaults to None
- **`apikey`** – API key to use. Defaults to None.
- **`bugzilla_url`** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`get`(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **`bug_number`** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

`put`(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **`bug`** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields (**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords (**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search ()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe (*start*, *end*)
When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.`**`Search`**(*bugsy*)

This allows searching for bugs in Bugzilla

`__init__`(*bugsy*)

Initialises the search object

Parameters **`bugsy`** – Bugsy instance to use to connect to Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`assigned_to`(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

`bug_number`(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **`bug_numbers`** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

`change_history_fields`(*fields*, *value=None*)

`include_fields`(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

`keywords`(**args*)

When search() is called it will search for the keywords passed in here

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

`search`()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.Bugsy`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields (**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords (**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search ()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe (*start*, *end*)
When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.`**`Search`**(*bugsy*)

This allows searching for bugs in Bugzilla

`__init__`(*bugsy*)

Initialises the search object

Parameters **`bugsy`** – Bugsy instance to use to connect to Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`assigned_to`(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

`bug_number`(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **`bug_numbers`** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

`change_history_fields`(*fields*, *value=None*)

`include_fields`(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

`keywords`(**args*)

When search() is called it will search for the keywords passed in here

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

`search`()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**`Bugsy`**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

`__init__`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **`username`** – Username to login with. Defaults to None
- **`password`** – Password to login with. Defaults to None
- **`userid`** – User ID to login with. Defaults to None
- **`cookie`** – Cookie to login with. Defaults to None
- **`apikey`** – API key to use. Defaults to None.
- **`bugzilla_url`** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`get`(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **`bug_number`** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

`put`(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **`bug`** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields (**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords (**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search ()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe (*start*, *end*)
When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.`**`Search`**(*bugsy*)

This allows searching for bugs in Bugzilla

`__init__`(*bugsy*)

Initialises the search object

Parameters **`bugsy`** – Bugsy instance to use to connect to Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`assigned_to`(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

`bug_number`(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **`bug_numbers`** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

`change_history_fields`(*fields*, *value=None*)

`include_fields`(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

`keywords`(**args*)

When search() is called it will search for the keywords passed in here

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

`search`()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.Bugsy`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields (**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords (**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search ()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe (*start*, *end*)
When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**`Bugsy`**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

`__init__`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`get`(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.`**`Search`**(*bugsy*)

This allows searching for bugs in Bugzilla

`__init__`(*bugsy*)

Initialises the search object

Parameters **`bugsy`** – Bugsy instance to use to connect to Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`assigned_to`(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

`bug_number`(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **`bug_numbers`** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

`change_history_fields`(*fields*, *value=None*)

`include_fields`(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

`keywords`(**args*)

When search() is called it will search for the keywords passed in here

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

`search`()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**`Bugsy`**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

`__init__`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **`username`** – Username to login with. Defaults to None
- **`password`** – Password to login with. Defaults to None
- **`userid`** – User ID to login with. Defaults to None
- **`cookie`** – Cookie to login with. Defaults to None
- **`apikey`** – API key to use. Defaults to None.
- **`bugzilla_url`** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`get`(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **`bug_number`** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

`put`(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **`bug`** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**`Bugsy`**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

`__init__`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`get`(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.`**`Search`**(*bugsy*)

This allows searching for bugs in Bugzilla

`__init__`(*bugsy*)

Initialises the search object

Parameters **`bugsy`** – Bugsy instance to use to connect to Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`assigned_to`(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

`bug_number`(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **`bug_numbers`** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

`change_history_fields`(*fields*, *value=None*)

`include_fields`(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

`keywords`(**args*)

When search() is called it will search for the keywords passed in here

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

`search`()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.Bugsy`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields (**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords (**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search ()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe (*start*, *end*)
When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.`**`Search`**(*bugsy*)

This allows searching for bugs in Bugzilla

`__init__`(*bugsy*)

Initialises the search object

Parameters **`bugsy`** – Bugsy instance to use to connect to Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`assigned_to`(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

`bug_number`(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **`bug_numbers`** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

`change_history_fields`(*fields*, *value=None*)

`include_fields`(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

`keywords`(**args*)

When search() is called it will search for the keywords passed in here

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

`search`()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                .keywords("checkin-needed")\
...                .include_fields("flags")\
...                .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields (**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords (**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search ()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe (*start*, *end*)
When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.`**`Search`**(*bugsy*)

This allows searching for bugs in Bugzilla

`__init__`(*bugsy*)

Initialises the search object

Parameters **`bugsy`** – Bugsy instance to use to connect to Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`assigned_to`(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

`bug_number`(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **`bug_numbers`** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

`change_history_fields`(*fields*, *value=None*)

`include_fields`(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

`keywords`(**args*)

When search() is called it will search for the keywords passed in here

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

`search`()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**`Bugsy`**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

`__init__`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **`username`** – Username to login with. Defaults to None
- **`password`** – Password to login with. Defaults to None
- **`userid`** – User ID to login with. Defaults to None
- **`cookie`** – Cookie to login with. Defaults to None
- **`apikey`** – API key to use. Defaults to None.
- **`bugzilla_url`** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`get`(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **`bug_number`** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

`put`(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **`bug`** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields (**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords (**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search ()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe (*start*, *end*)
When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**`Bugsy`**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

`__init__`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`get`(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.`**`Search`**(*bugsy*)

This allows searching for bugs in Bugzilla

`__init__`(*bugsy*)

Initialises the search object

Parameters **`bugsy`** – Bugsy instance to use to connect to Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`assigned_to`(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

`bug_number`(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **`bug_numbers`** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

`change_history_fields`(*fields*, *value=None*)

`include_fields`(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

`keywords`(**args*)

When search() is called it will search for the keywords passed in here

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

`search`()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.Bugsy`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields (**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords (**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search ()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe (*start*, *end*)
When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.`**`Search`**(*bugsy*)

This allows searching for bugs in Bugzilla

`__init__`(*bugsy*)

Initialises the search object

Parameters **`bugsy`** – Bugsy instance to use to connect to Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`assigned_to`(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

`bug_number`(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **`bug_numbers`** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

`change_history_fields`(*fields*, *value=None*)

`include_fields`(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

`keywords`(**args*)

When search() is called it will search for the keywords passed in here

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

`search`()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.Bugsy`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields (**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords (**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search ()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe (*start*, *end*)
When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.`**`Search`**(*bugsy*)

This allows searching for bugs in Bugzilla

`__init__`(*bugsy*)

Initialises the search object

Parameters **`bugsy`** – Bugsy instance to use to connect to Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`assigned_to`(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

`bug_number`(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **`bug_numbers`** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

`change_history_fields`(*fields*, *value=None*)

`include_fields`(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

`keywords`(**args*)

When search() is called it will search for the keywords passed in here

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

`search`()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**`Bugsy`**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

`__init__`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **`username`** – Username to login with. Defaults to None
- **`password`** – Password to login with. Defaults to None
- **`userid`** – User ID to login with. Defaults to None
- **`cookie`** – Cookie to login with. Defaults to None
- **`apikey`** – API key to use. Defaults to None.
- **`bugzilla_url`** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`get`(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **`bug_number`** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

`put`(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **`bug`** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                .keywords("checkin-needed")\
...                .include_fields("flags")\
...                .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields (**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords (**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search ()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe (*start*, *end*)
When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.`**`Search`**(*bugsy*)

This allows searching for bugs in Bugzilla

`__init__`(*bugsy*)

Initialises the search object

Parameters **`bugsy`** – Bugsy instance to use to connect to Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`assigned_to`(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

`bug_number`(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **`bug_numbers`** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

`change_history_fields`(*fields*, *value=None*)

`include_fields`(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

`keywords`(**args*)

When search() is called it will search for the keywords passed in here

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

`search`()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.Bugsy`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields (**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords (**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search ()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe (*start*, *end*)
When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.`**`Search`**(*bugsy*)

This allows searching for bugs in Bugzilla

`__init__`(*bugsy*)

Initialises the search object

Parameters **`bugsy`** – Bugsy instance to use to connect to Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`assigned_to`(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

`bug_number`(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **`bug_numbers`** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

`change_history_fields`(*fields*, *value=None*)

`include_fields`(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

`keywords`(**args*)

When search() is called it will search for the keywords passed in here

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

`search`()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**`Bugsy`**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

`__init__`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **`username`** – Username to login with. Defaults to None
- **`password`** – Password to login with. Defaults to None
- **`userid`** – User ID to login with. Defaults to None
- **`cookie`** – Cookie to login with. Defaults to None
- **`apikey`** – API key to use. Defaults to None.
- **`bugzilla_url`** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`get`(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **`bug_number`** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

`put`(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **`bug`** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields (**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords (**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search ()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe (*start*, *end*)
When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.`**`Search`**(*bugsy*)

This allows searching for bugs in Bugzilla

`__init__`(*bugsy*)

Initialises the search object

Parameters **`bugsy`** – Bugsy instance to use to connect to Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`assigned_to`(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

`bug_number`(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **`bug_numbers`** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

`change_history_fields`(*fields*, *value=None*)

`include_fields`(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

`keywords`(**args*)

When search() is called it will search for the keywords passed in here

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

`search`()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.Bugsy`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put(*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()
Obtain comments for this bug.

Returns a list of Comment instances.

id
Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform
Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)

Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)

Add tags to the comments

attachment_id

If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author

Return the login name of the comment's author.

bug_id

Return the ID of the bug that this comment is on.

creation_time

Return the time (in Bugzilla's timezone) that the comment was added.

creator

Return the login name of the comment's author.

id

Return the comment id that is associated with Bugzilla.

is_private

Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)

Add tags to the comments

tags

Return a set of comment tags currently set for the comment.

text

Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time

This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)

This allows searching for bugs in Bugzilla

__init__(*bugsy*)

Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS
Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)
Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__
list of weak references to the object (if defined)

add_comment (*comment*)
Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns `Search`

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__

list of weak references to the object (if defined)

get (*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug**(*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product
Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class bugsy.**Search**(*bugsy*)
This allows searching for bugs in Bugzilla

__init__(*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to(**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number (*bug_numbers*)
When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields (*fields*, *value=None*)

include_fields (**args*)
Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords (**args*)
When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search ()
Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe (*start*, *end*)
When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard (**args*)
When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

Indices and tables

- genindex
- modindex
- search

Bugsy

class `bugsy.`**`Bugsy`**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

`__init__`(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None
- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`get`(*bug_number*)

Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)

This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)

Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)

If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)

If a username and password are passed in but we don't receive a token then this error will be raised.

Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)

This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__ (*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment (*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution

Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status

Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary

Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()

Return the raw dict that is used inside this object

update()

Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

Comment Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags(*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

Search Changed in version 0.2.

class `bugsy.`**`Search`**(*bugsy*)

This allows searching for bugs in Bugzilla

`__init__`(*bugsy*)

Initialises the search object

Parameters **`bugsy`** – Bugsy instance to use to connect to Bugzilla.

`__weakref__`

list of weak references to the object (if defined)

`assigned_to`(**args*)

When search() is called it will search for bugs assigned to these users

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

`bug_number`(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **`bug_numbers`** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

`change_history_fields`(*fields*, *value=None*)

`include_fields`(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

`keywords`(**args*)

When search() is called it will search for the keywords passed in here

Parameters **`args`** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

`search`()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

2.4.2 Indices and tables

- genindex
- modindex
- search

2.4.3 Bugsy

class bugsy.**Bugsy**(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Bugsy allows easy getting and putting of Bugzilla bugs

__init__(*username=None*, *password=None*, *userid=None*, *cookie=None*, *api_key=None*, *bugzilla_url='https://bugzilla.mozilla.org/rest'*)

Initialises a new instance of Bugsy

Parameters

- **username** – Username to login with. Defaults to None
- **password** – Password to login with. Defaults to None
- **userid** – User ID to login with. Defaults to None
- **cookie** – Cookie to login with. Defaults to None

- **apikey** – API key to use. Defaults to None.
- **bugzilla_url** – URL endpoint to interact with. Defaults to

https://bugzilla.mozilla.org/rest

If a api_key is passed in, Bugsy will use this for authenticating requests. While not required to perform requests, if a username is passed in along with api_key, we will validate that the api key is valid for this username. Otherwise the api key is blindly used later.

If a username AND password are passed in Bugsy will try get a login token from Bugzilla. If we can't login then a LoginException will be raised.

If a userid AND cookie are passed in Bugsy will create a login token from them. If no username was passed in it will then try to get the username from Bugzilla.

__weakref__
list of weak references to the object (if defined)

get (*bug_number*)
Get a bug from Bugzilla. If there is a login token created during object initialisation it will be part of the query string passed to Bugzilla

Parameters **bug_number** – Bug Number that will be searched. If found will return a Bug object.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
```

put (*bug*)
This method allows you to create or update a bug on Bugzilla. You will have had to pass in a valid username and password to the object initialisation and recieved back a token.

Parameters **bug** – A Bug object either created by hand or by using get()

If there is no valid token then a BugsyException will be raised. If the object passed in is not a Bug then a BugsyException will be raised.

```
>>> bugzilla = Bugsy()
>>> bug = bugzilla.get(123456)
>>> bug.summary = "I like cheese and sausages"
>>> bugzilla.put(bug)
```

request (*path*, *method='GET'*, ***kwargs*)
Perform a HTTP request.

Given a relative Bugzilla URL path, an optional request method, and arguments suitable for requests.Request(), perform a HTTP request.

class bugsy.**BugsyException** (*msg*)
If while interacting with Bugzilla and we try do something that is not supported this error will be raised.

class bugsy.**LoginException** (*msg*)
If a username and password are passed in but we don't receive a token then this error will be raised.

2.4.4 Bug

class bugsy.**Bug** (*bugsy=None*, ***kwargs*)
This represents a Bugzilla Bug

OS

Property for getting or setting the OS that the bug occured on

```
>>> bug.OS
"All"
```

__init__(*bugsy=None*, ***kwargs*)

Defaults are set if there are no kwargs passed in. To pass in a dict create the Bug object like the following

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

```
>>> bug = Bug(**myDict)
```

__weakref__

list of weak references to the object (if defined)

add_comment(*comment*)

Adds a comment to a bug. If the bug object does not have a bug ID (ie you are creating a bug) then you will need to also call *put* on the `Bugsy` class.

```
>>> bug.add_comment("I like sausages")
>>> bugzilla.put(bug)
```

If it does have a bug id then this will immediately post to the server

```
>>> bug.add_comment("I like eggs too")
```

More examples can be found at: https://github.com/AutomatedTester/Bugsy/blob/master/example/add_comments.py

assigned_to

Property for getting the bug assignee

```
>>> bug.assigned_to
"automatedtester@mozilla.com"
```

component

Property for getting the bug component

```
>>> bug.component
General
```

get_comments()

Obtain comments for this bug.

Returns a list of Comment instances.

id

Property for getting the ID of a bug.

```
>>> bug.id
123456
```

platform

Property for getting the bug platform

```
>>> bug.platform
"ARM"
```

product

Property for getting the bug product

```
>>> bug.product
Core
```

resolution
Property for getting or setting the bug resolution

```
>>> bug.resolution
"FIXED"
```

status
Property for getting or setting the bug status

```
>>> bug.status
"REOPENED"
```

summary
Property for getting and setting the bug summary

```
>>> bug.summary
"I like cheese"
```

to_dict()
Return the raw dict that is used inside this object

update()
Update this object with the latest changes from Bugzilla

```
>>> bug.status
'NEW'
#Changes happen on Bugzilla
>>> bug.update()
>>> bug.status
'FIXED'
```

version
Property for getting the bug platform

```
>>> bug.version
"TRUNK"
```

class bugsy.**BugException**(*msg*)
If we try do something that is not allowed to a bug then this error is raised

2.4.5 Comment

Changed in version 0.3.

class bugsy.**Comment**(*bugsy=None*, ***kwargs*)
Represents a single Bugzilla comment.

To get comments you need to do the following

```
>>> bugs = bugzilla.search_for.keywords("checkin-needed").search()
>>> comments = bugs[0].get_comments()
>>> # Returns the comment 0 of the first checkin-needed bug
>>> comments[0].text
```

add_tags(*tags*)
Add tags to the comments

attachment_id
If the comment was made on an attachment, return the ID of that attachment. Otherwise it will return None.

author
Return the login name of the comment's author.

bug_id
Return the ID of the bug that this comment is on.

creation_time
Return the time (in Bugzilla's timezone) that the comment was added.

creator
Return the login name of the comment's author.

id
Return the comment id that is associated with Bugzilla.

is_private
Return True if this comment is private (only visible to a certain group called the "insidergroup").

remove_tags (*tags*)
Add tags to the comments

tags
Return a set of comment tags currently set for the comment.

text
Return the text that is in this comment

```
>>> comment.text # David really likes cheese apparently
```

time
This is exactly same as `creation_time`.

For compatibility, time is still usable. However, please note that time may be deprecated and removed in a future release.

Prefer `creation_time` instead.

2.4.6 Search

Changed in version 0.2.

class bugsy.**Search** (*bugsy*)
This allows searching for bugs in Bugzilla

__init__ (*bugsy*)
Initialises the search object

Parameters **bugsy** – Bugsy instance to use to connect to Bugzilla.

__weakref__
list of weak references to the object (if defined)

assigned_to (**args*)
When search() is called it will search for bugs assigned to these users

Parameters **args** – items passed in will be turned into a list

Returns `Search`

```
>>> bugzilla.search_for.assigned_to("dburns@mozilla.com")
```

bug_number(*bug_numbers*)

When you want to search for a bugs and be able to change the fields returned.

Parameters **bug_numbers** – A string for the bug number or a list of strings

Returns *Search*

```
>>> bugzilla.search_for.bug_number(['123123', '123456'])
```

change_history_fields(*fields*, *value=None*)

include_fields(**args*)

Include fields is the fields that you want to be returned when searching. These are in addition to the fields that are always included below.

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.include_fields("flags")
```

The following fields are always included in search: 'version', 'id', 'summary', 'status', 'op_sys', 'resolution', 'product', 'component', 'platform'

keywords(**args*)

When search() is called it will search for the keywords passed in here

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.keywords("checkin-needed")
```

search()

Call the Bugzilla endpoint that will do the search. It will take the information used in other methods on the Search object and build up the query string. If no bugs are found then an empty list is returned.

```
>>> bugs = bugzilla.search_for\
...                 .keywords("checkin-needed")\
...                 .include_fields("flags")\
...                 .search()
```

summary(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.summary("663399")
```

timeframe(*start*, *end*)

When you want to search bugs for a certain time frame.

Parameters

- **start** –
- **end** –

Returns *Search*

whiteboard(**args*)

When search is called it will search for bugs with the words passed into the methods

Parameters **args** – items passed in will be turned into a list

Returns *Search*

```
>>> bugzilla.search_for.whiteboard("affects")
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

b

bugsy, 616

Symbols

__init__() (bugsy.Bug method), 127, 133, 138, 144, 149, 154, 160, 165, 171, 176, 181, 187, 192, 198, 203, 208, 214, 219, 225, 230, 235, 241, 246, 252, 257, 262, 268, 273, 279, 284, 289, 295, 300, 306, 311, 316, 322, 327, 333, 338, 343, 349, 354, 360, 365, 370, 376, 381, 387, 392, 397, 403, 408, 414, 419, 424, 430, 435, 441, 446, 451, 457, 462, 468, 473, 478, 484, 489, 495, 500, 505, 511, 516, 522, 527, 532, 538, 543, 549, 554, 559, 565, 570, 576, 581, 586, 592, 597, 603, 608, 614

__init__() (bugsy.Bugsy method), 126, 132, 137, 142, 148, 153, 159, 164, 169, 175, 180, 186, 191, 196, 202, 207, 213, 218, 223, 229, 234, 240, 245, 250, 256, 261, 267, 272, 277, 283, 288, 294, 299, 304, 310, 315, 321, 326, 331, 337, 342, 348, 353, 358, 364, 369, 375, 380, 385, 391, 396, 402, 407, 412, 418, 423, 429, 434, 439, 445, 450, 456, 461, 466, 472, 477, 483, 488, 493, 499, 504, 510, 515, 520, 526, 531, 537, 542, 547, 553, 558, 564, 569, 574, 580, 585, 591, 596, 601, 607, 612

__init__() (bugsy.Search method), 130, 135, 141, 146, 152, 157, 162, 168, 173, 179, 184, 189, 195, 200, 206, 211, 216, 222, 227, 233, 238, 243, 249, 254, 260, 265, 270, 276, 281, 287, 292, 297, 303, 308, 314, 319, 324, 330, 335, 341, 346, 351, 357, 362, 368, 373, 378, 384, 389, 395, 400, 405, 411, 416, 422, 427, 432, 438, 443, 449, 454, 459, 465, 470, 476, 481, 486, 492, 497, 503, 508, 513, 519, 524, 530, 535, 540, 546, 551, 557, 562, 567, 573, 578, 584, 589, 594, 600, 605, 611, 616

__weakref__ (bugsy.Bug attribute), 128, 133, 138, 144, 149, 155, 160, 165, 171, 176, 182, 187, 192, 198, 203, 209, 214, 219, 225, 230, 236, 241, 246, 252, 257, 263, 268, 273, 279, 284, 290, 295, 300, 306, 311, 317, 322, 327, 333, 338, 344, 349, 354, 360, 365, 371, 376, 381, 387, 392, 398, 403, 408, 414, 419, 425, 430, 435, 441, 446, 452, 457, 462, 468, 473, 479, 484, 489, 495, 500, 506, 511, 516, 522, 527, 533, 538, 543, 549, 554, 560, 565, 570, 576, 581, 587, 592, 597, 603, 608, 614

__weakref__ (bugsy.Bugsy attribute), 127, 132, 137, 143, 148, 154, 159, 164, 170, 175, 181, 186, 191, 197, 202, 208, 213, 218, 224, 229, 235, 240, 245, 251, 256, 262, 267, 272, 278, 283, 289, 294, 299, 305, 310, 316, 321, 326, 332, 337, 343, 348, 353, 359, 364, 370, 375, 380, 386, 391, 397, 402, 407, 413, 418, 424, 429, 434, 440, 445, 451, 456, 461, 467, 472, 478, 483, 488, 494, 499, 505, 510, 515, 521, 526, 532, 537, 542, 548, 553, 559, 564, 569, 575, 580, 586, 591, 596, 602, 607, 613

__weakref__ (bugsy.Search attribute), 130, 136, 141, 146, 152, 157, 163, 168, 173, 179, 184, 190, 195, 200, 206, 211, 217, 222, 227, 233, 238, 244, 249, 254, 260, 265, 271, 276, 281, 287, 292, 298, 303, 308, 314, 319, 325, 330, 335, 341, 346, 352, 357, 362, 368, 373, 379, 384, 389, 395, 400, 406, 411, 416, 422, 427, 433, 438, 443, 449, 454, 460, 465, 470, 476, 481, 487, 492, 497, 503, 508, 514, 519, 524, 530, 535, 541, 546, 551, 557, 562, 568, 573, 578, 584, 589, 595, 600, 605, 611, 616

A

add_comment() (bugsy.Bug method), 128, 133, 138, 144, 149, 155, 160, 165, 171, 176, 182, 187, 192, 198, 203, 209, 214, 219, 225, 230, 236, 241, 246, 252, 257, 263, 268, 273, 279, 284, 290, 295, 300, 306, 311, 317, 322, 327, 333, 338, 344, 349, 354, 360, 365, 371, 376, 381, 387, 392, 398, 403, 408, 414, 419, 425, 430, 435, 441, 446, 452, 457, 462, 468, 473, 479, 484, 489, 495, 500, 506, 511, 516, 522, 527, 533, 538, 543, 549, 554, 560, 565, 570, 576, 581, 587, 592, 597, 603, 608, 614

add_tags() (bugsy.Comment method), 129, 135, 140, 145,

151, 156, 162, 167, 172, 178, 183, 189, 194, 199, 205, 210, 216, 221, 226, 232, 237, 243, 248, 253, 259, 264, 270, 275, 280, 286, 291, 297, 302, 307, 313, 318, 324, 329, 334, 340, 345, 351, 356, 361, 367, 372, 378, 383, 388, 394, 399, 405, 410, 415, 421, 426, 432, 437, 442, 448, 453, 459, 464, 469, 475, 480, 486, 491, 496, 502, 507, 513, 518, 523, 529, 534, 540, 545, 550, 556, 561, 567, 572, 577, 583, 588, 594, 599, 604, 610, 615

assigned_to (bugsy.Bug attribute), 128, 133, 139, 144, 149, 155, 160, 166, 171, 176, 182, 187, 193, 198, 203, 209, 214, 220, 225, 230, 236, 241, 247, 252, 257, 263, 268, 274, 279, 284, 290, 295, 301, 306, 311, 317, 322, 328, 333, 338, 344, 349, 355, 360, 365, 371, 376, 382, 387, 392, 398, 403, 409, 414, 419, 425, 430, 436, 441, 446, 452, 457, 463, 468, 473, 479, 484, 490, 495, 500, 506, 511, 517, 522, 527, 533, 538, 544, 549, 554, 560, 565, 571, 576, 581, 587, 592, 598, 603, 608, 614

assigned_to() (bugsy.Search method), 130, 136, 141, 146, 152, 157, 163, 168, 173, 179, 184, 190, 195, 200, 206, 211, 217, 222, 227, 233, 238, 244, 249, 254, 260, 265, 271, 276, 281, 287, 292, 298, 303, 308, 314, 319, 325, 330, 335, 341, 346, 352, 357, 362, 368, 373, 379, 384, 389, 395, 400, 406, 411, 416, 422, 427, 433, 438, 443, 449, 454, 460, 465, 470, 476, 481, 487, 492, 497, 503, 508, 514, 519, 524, 530, 535, 541, 546, 551, 557, 562, 568, 573, 578, 584, 589, 595, 600, 605, 611, 616

attachment_id (bugsy.Comment attribute), 129, 135, 140, 145, 151, 156, 162, 167, 172, 178, 183, 189, 194, 199, 205, 210, 216, 221, 226, 232, 237, 243, 248, 253, 259, 264, 270, 275, 280, 286, 291, 297, 302, 307, 313, 318, 324, 329, 334, 340, 345, 351, 356, 361, 367, 372, 378, 383, 388, 394, 399, 405, 410, 415, 421, 426, 432, 437, 442, 448, 453, 459, 464, 469, 475, 480, 486, 491, 496, 502, 507, 513, 518, 523, 529, 534, 540, 545, 550, 556, 561, 567, 572, 577, 583, 588, 594, 599, 604, 610, 615

author (bugsy.Comment attribute), 129, 135, 140, 146, 151, 156, 162, 167, 173, 178, 183, 189, 194, 200, 205, 210, 216, 221, 227, 232, 237, 243, 248, 254, 259, 264, 270, 275, 281, 286, 291, 297, 302, 308, 313, 318, 324, 329, 335, 340, 345, 351, 356, 362, 367, 372, 378, 383, 389, 394, 399, 405, 410, 416, 421, 426, 432, 437, 443, 448, 453, 459, 464, 470, 475, 480, 486, 491, 497, 502, 507, 513, 518, 524, 529, 534, 540, 545, 551, 556, 561, 567, 572, 578, 583, 588, 594, 599, 605, 610, 616

B

Bug (class in bugsy), 127, 133, 138, 144, 149, 154, 160, 165, 171, 176, 181, 187, 192, 198, 203, 208, 214, 219, 225, 230, 235, 241, 246, 252, 257, 262, 268, 273, 279, 284, 289, 295, 300, 306, 311, 316, 322, 327, 333, 338, 343, 349, 354, 360, 365, 370, 376, 381, 387, 392, 397, 403, 408, 414, 419, 424, 430, 435, 441, 446, 451, 457, 462, 468, 473, 478, 484, 489, 495, 500, 505, 511, 516, 522, 527, 532, 538, 543, 549, 554, 559, 565, 570, 576, 581, 586, 592, 597, 603, 608, 613

bug_id (bugsy.Comment attribute), 129, 135, 140, 146, 151, 156, 162, 167, 173, 178, 183, 189, 194, 200, 205, 210, 216, 221, 227, 232, 237, 243, 248, 254, 259, 264, 270, 275, 281, 286, 291, 297, 302, 308, 313, 318, 324, 329, 335, 340, 345, 351, 356, 362, 367, 372, 378, 383, 389, 394, 399, 405, 410, 416, 421, 426, 432, 437, 443, 448, 453, 459, 464, 470, 475, 480, 486, 491, 497, 502, 507, 513, 518, 524, 529, 534, 540, 545, 551, 556, 561, 567, 572, 578, 583, 588, 594, 599, 605, 610, 616

bug_number() (bugsy.Search method), 130, 136, 141, 146, 152, 157, 163, 168, 173, 179, 184, 190, 195, 200, 206, 211, 217, 222, 227, 233, 238, 244, 249, 254, 260, 265, 271, 276, 281, 287, 292, 298, 303, 308, 314, 319, 325, 330, 335, 341, 346, 352, 357, 362, 368, 373, 379, 384, 389, 395, 400, 406, 411, 416, 422, 427, 433, 438, 443, 449, 454, 460, 465, 470, 476, 481, 487, 492, 497, 503, 508, 514, 519, 524, 530, 535, 541, 546, 551, 557, 562, 568, 573, 578, 584, 589, 595, 600, 605, 611, 617

BugException (class in bugsy), 129, 134, 140, 145, 151, 156, 161, 167, 172, 178, 183, 188, 194, 199, 205, 210, 215, 221, 226, 232, 237, 242, 248, 253, 259, 264, 269, 275, 280, 286, 291, 296, 302, 307, 313, 318, 323, 329, 334, 340, 345, 350, 356, 361, 367, 372, 377, 383, 388, 394, 399, 404, 410, 415, 421, 426, 431, 437, 442, 448, 453, 458, 464, 469, 475, 480, 485, 491, 496, 502, 507, 512, 518, 523, 529, 534, 539, 545, 550, 556, 561, 566, 572, 577, 583, 588, 593, 599, 604, 610, 615

Bugsy (class in bugsy), 126, 132, 137, 142, 148, 153, 159, 164, 169, 175, 180, 186, 191, 196, 202, 207, 213, 218, 223, 229, 234, 240, 245, 250, 256, 261, 267, 272, 277, 283, 288, 294, 299, 304, 310, 315, 321, 326, 331, 337, 342, 348, 353, 358, 364, 369, 375, 380, 385, 391, 396, 402, 407, 412, 418, 423, 429, 434, 439, 445, 450, 456, 461, 466, 472, 477, 483, 488, 493, 499, 504, 510, 515, 520, 526, 531, 537, 542, 547,

553, 558, 564, 569, 574, 580, 585, 591, 596, 601, 607, 612
bugsy (module), 126, 127, 129, 130, 132, 133, 135, 137, 138, 140–142, 144–146, 148, 149, 151–154, 156, 157, 159, 160, 162, 164, 165, 167–169, 171–173, 175, 176, 178–181, 183, 184, 186, 187, 189, 191, 192, 194–196, 198–200, 202, 203, 205–208, 210, 211, 213, 214, 216, 218, 219, 221–223, 225–227, 229, 230, 232–235, 237, 238, 240, 241, 243, 245, 246, 248–250, 252–254, 256, 257, 259–262, 264, 265, 267, 268, 270, 272, 273, 275–277, 279–281, 283, 284, 286–289, 291, 292, 294, 295, 297, 299, 300, 302–304, 306–308, 310, 311, 313–316, 318, 319, 321, 322, 324, 326, 327, 329–331, 333–335, 337, 338, 340–343, 345, 346, 348, 349, 351, 353, 354, 356–358, 360–362, 364, 365, 367–370, 372, 373, 375, 376, 378, 380, 381, 383–385, 387–389, 391, 392, 394–397, 399, 400, 402, 403, 405, 407, 408, 410–412, 414–416, 418, 419, 421–424, 426, 427, 429, 430, 432, 434, 435, 437–439, 441–443, 445, 446, 448–451, 453, 454, 456, 457, 459, 461, 462, 464–466, 468–470, 472, 473, 475–478, 480, 481, 483, 484, 486, 488, 489, 491–493, 495–497, 499, 500, 502–505, 507, 508, 510, 511, 513, 515, 516, 518–520, 522–524, 526, 527, 529–532, 534, 535, 537, 538, 540, 542, 543, 545–547, 549–551, 553, 554, 556–559, 561, 562, 564, 565, 567, 569, 570, 572–574, 576–578, 580, 581, 583–586, 588, 589, 591, 592, 594, 596, 597, 599–601, 603–605, 607, 608, 610–613, 615, 616
BugsyException (class in bugsy), 127, 133, 138, 143, 149, 154, 160, 165, 170, 176, 181, 187, 192, 197, 203, 208, 214, 219, 224, 230, 235, 241, 246, 251, 257, 262, 268, 273, 278, 284, 289, 295, 300, 305, 311, 316, 322, 327, 332, 338, 343, 349, 354, 359, 365, 370, 376, 381, 386, 392, 397, 403, 408, 413, 419, 424, 430, 435, 440, 446, 451, 457, 462, 467, 473, 478, 484, 489, 494, 500, 505, 511, 516, 521, 527, 532, 538, 543, 548, 554, 559, 565, 570, 575, 581, 586, 592, 597, 602, 608, 613

C

change_history_fields() (bugsy.Search method), 130, 136, 141, 147, 152, 157, 163, 168, 174, 179, 184, 190, 195, 201, 206, 211, 217, 222, 228, 233, 238, 244, 249, 255, 260, 265, 271, 276, 282, 287, 292, 298, 303, 309, 314, 319, 325, 330, 336, 341, 346, 352, 357, 363, 368, 373, 379, 384, 390, 395, 400, 406, 411, 417, 422, 427, 433, 438, 444, 449, 454, 460, 465, 471, 476, 481, 487, 492, 498, 503, 508, 514, 519, 525, 530, 535, 541, 546, 552, 557, 562, 568, 573, 579, 584, 589, 595, 600, 606, 611, 617
Comment (class in bugsy), 129, 135, 140, 145, 151, 156, 162, 167, 172, 178, 183, 189, 194, 199, 205, 210, 216, 221, 226, 232, 237, 243, 248, 253, 259, 264, 270, 275, 280, 286, 291, 297, 302, 307, 313, 318, 324, 329, 334, 340, 345, 351, 356, 361, 367, 372, 378, 383, 388, 394, 399, 405, 410, 415, 421, 426, 432, 437, 442, 448, 453, 459, 464, 469, 475, 480, 486, 491, 496, 502, 507, 513, 518, 523, 529, 534, 540, 545, 550, 556, 561, 567, 572, 577, 583, 588, 594, 599, 604, 610, 615
component (bugsy.Bug attribute), 128, 133, 139, 144, 150, 155, 160, 166, 171, 177, 182, 187, 193, 198, 204, 209, 214, 220, 225, 231, 236, 241, 247, 252, 258, 263, 268, 274, 279, 285, 290, 295, 301, 306, 312, 317, 322, 328, 333, 339, 344, 349, 355, 360, 366, 371, 376, 382, 387, 393, 398, 403, 409, 414, 420, 425, 430, 436, 441, 447, 452, 457, 463, 468, 474, 479, 484, 490, 495, 501, 506, 511, 517, 522, 528, 533, 538, 544, 549, 555, 560, 565, 571, 576, 582, 587, 592, 598, 603, 609, 614
creation_time (bugsy.Comment attribute), 129, 135, 140, 146, 151, 156, 162, 167, 173, 178, 183, 189, 194, 200, 205, 210, 216, 221, 227, 232, 237, 243, 248, 254, 259, 264, 270, 275, 281, 286, 291, 297, 302, 308, 313, 318, 324, 329, 335, 340, 345, 351, 356, 362, 367, 372, 378, 383, 389, 394, 399, 405, 410, 416, 421, 426, 432, 437, 443, 448, 453, 459, 464, 470, 475, 480, 486, 491, 497, 502, 507, 513, 518, 524, 529, 534, 540, 545, 551, 556, 561, 567, 572, 578, 583, 588, 594, 599, 605, 610, 616
creator (bugsy.Comment attribute), 130, 135, 140, 146, 151, 157, 162, 167, 173, 178, 184, 189, 194, 200, 205, 211, 216, 221, 227, 232, 238, 243, 248, 254, 259, 265, 270, 275, 281, 286, 292, 297, 302, 308, 313, 319, 324, 329, 335, 340, 346, 351, 356, 362, 367, 373, 378, 383, 389, 394, 400, 405, 410, 416, 421, 427, 432, 437, 443, 448, 454, 459, 464, 470, 475, 481, 486, 491, 497, 502, 508, 513, 518, 524, 529, 535, 540, 545, 551, 556, 562, 567, 572, 578, 583, 589, 594, 599, 605, 610, 616

G

get() (bugsy.Bugsy method), 127, 132, 138, 143, 148, 154, 159, 165, 170, 175, 181, 186, 192, 197, 202, 208, 213, 219, 224, 229, 235, 240, 246, 251, 256, 262, 267, 273, 278, 283, 289, 294, 300, 305, 310, 316, 321, 327, 332, 337, 343,

348, 354, 359, 364, 370, 375, 381, 386, 391, 397, 402, 408, 413, 418, 424, 429, 435, 440, 445, 451, 456, 462, 467, 472, 478, 483, 489, 494, 499, 505, 510, 516, 521, 526, 532, 537, 543, 548, 553, 559, 564, 570, 575, 580, 586, 591, 597, 602, 607, 613

get_comments() (bugsy.Bug method), 128, 133, 139, 144, 150, 155, 160, 166, 171, 177, 182, 187, 193, 198, 204, 209, 214, 220, 225, 231, 236, 241, 247, 252, 258, 263, 268, 274, 279, 285, 290, 295, 301, 306, 312, 317, 322, 328, 333, 339, 344, 349, 355, 360, 366, 371, 376, 382, 387, 393, 398, 403, 409, 414, 420, 425, 430, 436, 441, 447, 452, 457, 463, 468, 474, 479, 484, 490, 495, 501, 506, 511, 517, 522, 528, 533, 538, 544, 549, 555, 560, 565, 571, 576, 582, 587, 592, 598, 603, 609, 614

I

id (bugsy.Bug attribute), 128, 134, 139, 144, 150, 155, 161, 166, 171, 177, 182, 188, 193, 198, 204, 209, 215, 220, 225, 231, 236, 242, 247, 252, 258, 263, 269, 274, 279, 285, 290, 296, 301, 306, 312, 317, 323, 328, 333, 339, 344, 350, 355, 360, 366, 371, 377, 382, 387, 393, 398, 404, 409, 414, 420, 425, 431, 436, 441, 447, 452, 458, 463, 468, 474, 479, 485, 490, 495, 501, 506, 512, 517, 522, 528, 533, 539, 544, 549, 555, 560, 566, 571, 576, 582, 587, 593, 598, 603, 609, 614

id (bugsy.Comment attribute), 130, 135, 140, 146, 151, 157, 162, 167, 173, 178, 184, 189, 194, 200, 205, 211, 216, 221, 227, 232, 238, 243, 248, 254, 259, 265, 270, 275, 281, 286, 292, 297, 302, 308, 313, 319, 324, 329, 335, 340, 346, 351, 356, 362, 367, 373, 378, 383, 389, 394, 400, 405, 410, 416, 421, 427, 432, 437, 443, 448, 454, 459, 464, 470, 475, 481, 486, 491, 497, 502, 508, 513, 518, 524, 529, 535, 540, 545, 551, 556, 562, 567, 572, 578, 583, 589, 594, 599, 605, 610, 616

include_fields() (bugsy.Search method), 130, 136, 141, 147, 152, 157, 163, 168, 174, 179, 184, 190, 195, 201, 206, 211, 217, 222, 228, 233, 238, 244, 249, 255, 260, 265, 271, 276, 282, 287, 292, 298, 303, 309, 314, 319, 325, 330, 336, 341, 346, 352, 357, 363, 368, 373, 379, 384, 390, 395, 400, 406, 411, 417, 422, 427, 433, 438, 444, 449, 454, 460, 465, 471, 476, 481, 487, 492, 498, 503, 508, 514, 519, 525, 530, 535, 541, 546, 552, 557, 562, 568, 573, 579, 584, 589, 595, 600, 606, 611, 617

is_private (bugsy.Comment attribute), 130, 135, 140, 146, 151, 157, 162, 167, 173, 178, 184, 189, 194, 200, 205, 211, 216, 221, 227, 232, 238, 243, 248, 254, 259, 265, 270, 275, 281, 286, 292, 297, 302, 308, 313, 319, 324, 329, 335, 340, 346, 351, 356, 362, 367, 373, 378, 383, 389, 394, 400, 405, 410, 416, 421, 427, 432, 437, 443, 448, 454, 459, 464, 470, 475, 481, 486, 491, 497, 502, 508, 513, 518, 524, 529, 535, 540, 545, 551, 556, 562, 567, 572, 578, 583, 589, 594, 599, 605, 610, 616

K

keywords() (bugsy.Search method), 131, 136, 141, 147, 152, 158, 163, 168, 174, 179, 185, 190, 195, 201, 206, 212, 217, 222, 228, 233, 239, 244, 249, 255, 260, 266, 271, 276, 282, 287, 293, 298, 303, 309, 314, 320, 325, 330, 336, 341, 347, 352, 357, 363, 368, 374, 379, 384, 390, 395, 401, 406, 411, 417, 422, 428, 433, 438, 444, 449, 455, 460, 465, 471, 476, 482, 487, 492, 498, 503, 509, 514, 519, 525, 530, 536, 541, 546, 552, 557, 563, 568, 573, 579, 584, 590, 595, 600, 606, 611, 617

L

LoginException (class in bugsy), 127, 133, 138, 143, 149, 154, 160, 165, 170, 176, 181, 187, 192, 197, 203, 208, 214, 219, 224, 230, 235, 241, 246, 251, 257, 262, 268, 273, 278, 284, 289, 295, 300, 305, 311, 316, 322, 327, 332, 338, 343, 349, 354, 359, 365, 370, 376, 381, 386, 392, 397, 403, 408, 413, 419, 424, 430, 435, 440, 446, 451, 457, 462, 467, 473, 478, 484, 489, 494, 500, 505, 511, 516, 521, 527, 532, 538, 543, 548, 554, 559, 565, 570, 575, 581, 586, 592, 597, 602, 608, 613

O

OS (bugsy.Bug attribute), 127, 133, 138, 144, 149, 154, 160, 165, 171, 176, 181, 187, 192, 198, 203, 208, 214, 219, 225, 230, 235, 241, 246, 252, 257, 262, 268, 273, 279, 284, 289, 295, 300, 306, 311, 316, 322, 327, 333, 338, 343, 349, 354, 360, 365, 370, 376, 381, 387, 392, 397, 403, 408, 414, 419, 424, 430, 435, 441, 446, 451, 457, 462, 468, 473, 478, 484, 489, 495, 500, 505, 511, 516, 522, 527, 532, 538, 543, 549, 554, 559, 565, 570, 576, 581, 586, 592, 597, 603, 608, 613

P

platform (bugsy.Bug attribute), 128, 134, 139, 144, 150, 155, 161, 166, 171, 177, 182, 188, 193, 198, 204, 209, 215, 220, 225, 231, 236, 242, 247, 252, 258, 263, 269, 274, 279, 285, 290, 296,

301, 306, 312, 317, 323, 328, 333, 339, 344, 350, 355, 360, 366, 371, 377, 382, 387, 393, 398, 404, 409, 414, 420, 425, 431, 436, 441, 447, 452, 458, 463, 468, 474, 479, 485, 490, 495, 501, 506, 512, 517, 522, 528, 533, 539, 544, 549, 555, 560, 566, 571, 576, 582, 587, 593, 598, 603, 609, 614

product (bugsy.Bug attribute), 128, 134, 139, 144, 150, 155, 161, 166, 171, 177, 182, 188, 193, 198, 204, 209, 215, 220, 225, 231, 236, 242, 247, 252, 258, 263, 269, 274, 279, 285, 290, 296, 301, 306, 312, 317, 323, 328, 333, 339, 344, 350, 355, 360, 366, 371, 377, 382, 387, 393, 398, 404, 409, 414, 420, 425, 431, 436, 441, 447, 452, 458, 463, 468, 474, 479, 485, 490, 495, 501, 506, 512, 517, 522, 528, 533, 539, 544, 549, 555, 560, 566, 571, 576, 582, 587, 593, 598, 603, 609, 614

put() (bugsy.Bugsy method), 127, 132, 138, 143, 148, 154, 159, 165, 170, 175, 181, 186, 192, 197, 202, 208, 213, 219, 224, 229, 235, 240, 246, 251, 256, 262, 267, 273, 278, 283, 289, 294, 300, 305, 310, 316, 321, 327, 332, 337, 343, 348, 354, 359, 364, 370, 375, 381, 386, 391, 397, 402, 408, 413, 418, 424, 429, 435, 440, 445, 451, 456, 462, 467, 472, 478, 483, 489, 494, 499, 505, 510, 516, 521, 526, 532, 537, 543, 548, 553, 559, 564, 570, 575, 580, 586, 591, 597, 602, 607, 613

R

remove_tags() (bugsy.Comment method), 130, 135, 140, 146, 151, 157, 162, 167, 173, 178, 184, 189, 194, 200, 205, 211, 216, 221, 227, 232, 238, 243, 248, 254, 259, 265, 270, 275, 281, 286, 292, 297, 302, 308, 313, 319, 324, 329, 335, 340, 346, 351, 356, 362, 367, 373, 378, 383, 389, 394, 400, 405, 410, 416, 421, 427, 432, 437, 443, 448, 454, 459, 464, 470, 475, 481, 486, 491, 497, 502, 508, 513, 518, 524, 529, 535, 540, 545, 551, 556, 562, 567, 572, 578, 583, 589, 594, 599, 605, 610, 616

request() (bugsy.Bugsy method), 127, 133, 138, 143, 149, 154, 160, 165, 170, 176, 181, 187, 192, 197, 203, 208, 214, 219, 224, 230, 235, 241, 246, 251, 257, 262, 268, 273, 278, 284, 289, 295, 300, 305, 311, 316, 322, 327, 332, 338, 343, 349, 354, 359, 365, 370, 376, 381, 386, 392, 397, 403, 408, 413, 419, 424, 430, 435, 440, 446, 451, 457, 462, 467, 473, 478, 484, 489, 494, 500, 505, 511, 516, 521, 527, 532, 538, 543, 548, 554, 559, 565, 570, 575, 581, 586, 592, 597, 602, 608, 613

resolution (bugsy.Bug attribute), 128, 134, 139, 145, 150, 155, 161, 166, 172, 177, 182, 188, 193, 199, 204, 209, 215, 220, 226, 231, 236, 242, 247, 253, 258, 263, 269, 274, 280, 285, 290, 296, 301, 307, 312, 317, 323, 328, 334, 339, 344, 350, 355, 361, 366, 371, 377, 382, 388, 393, 398, 404, 409, 415, 420, 425, 431, 436, 442, 447, 452, 458, 463, 469, 474, 479, 485, 490, 496, 501, 506, 512, 517, 523, 528, 533, 539, 544, 550, 555, 560, 566, 571, 577, 582, 587, 593, 598, 604, 609, 615

S

Search (class in bugsy), 130, 135, 141, 146, 152, 157, 162, 168, 173, 179, 184, 189, 195, 200, 206, 211, 216, 222, 227, 233, 238, 243, 249, 254, 260, 265, 270, 276, 281, 287, 292, 297, 303, 308, 314, 319, 324, 330, 335, 341, 346, 351, 357, 362, 368, 373, 378, 384, 389, 395, 400, 405, 411, 416, 422, 427, 432, 438, 443, 449, 454, 459, 465, 470, 476, 481, 486, 492, 497, 503, 508, 513, 519, 524, 530, 535, 540, 546, 551, 557, 562, 567, 573, 578, 584, 589, 594, 600, 605, 611, 616

search() (bugsy.Search method), 131, 136, 142, 147, 152, 158, 163, 169, 174, 179, 185, 190, 196, 201, 206, 212, 217, 223, 228, 233, 239, 244, 250, 255, 260, 266, 271, 277, 282, 287, 293, 298, 304, 309, 314, 320, 325, 331, 336, 341, 347, 352, 358, 363, 368, 374, 379, 385, 390, 395, 401, 406, 412, 417, 422, 428, 433, 439, 444, 449, 455, 460, 466, 471, 476, 482, 487, 493, 498, 503, 509, 514, 520, 525, 530, 536, 541, 547, 552, 557, 563, 568, 574, 579, 584, 590, 595, 601, 606, 611, 617

status (bugsy.Bug attribute), 129, 134, 139, 145, 150, 156, 161, 166, 172, 177, 183, 188, 193, 199, 204, 210, 215, 220, 226, 231, 237, 242, 247, 253, 258, 264, 269, 274, 280, 285, 291, 296, 301, 307, 312, 318, 323, 328, 334, 339, 345, 350, 355, 361, 366, 372, 377, 382, 388, 393, 399, 404, 409, 415, 420, 426, 431, 436, 442, 447, 453, 458, 463, 469, 474, 480, 485, 490, 496, 501, 507, 512, 517, 523, 528, 534, 539, 544, 550, 555, 561, 566, 571, 577, 582, 588, 593, 598, 604, 609, 615

summary (bugsy.Bug attribute), 129, 134, 139, 145, 150, 156, 161, 166, 172, 177, 183, 188, 193, 199, 204, 210, 215, 220, 226, 231, 237, 242, 247, 253, 258, 264, 269, 274, 280, 285, 291, 296, 301, 307, 312, 318, 323, 328, 334, 339, 345, 350, 355, 361, 366, 372, 377, 382, 388, 393, 399, 404, 409, 415, 420, 426, 431, 436, 442, 447, 453, 458, 463, 469, 474, 480, 485, 490, 496, 501, 507, 512, 517, 523, 528, 534, 539,

544, 550, 555, 561, 566, 571, 577, 582, 588, 593, 598, 604, 609, 615
summary() (bugsy.Search method), 131, 136, 142, 147, 153, 158, 163, 169, 174, 180, 185, 190, 196, 201, 207, 212, 217, 223, 228, 234, 239, 244, 250, 255, 261, 266, 271, 277, 282, 288, 293, 298, 304, 309, 315, 320, 325, 331, 336, 342, 347, 352, 358, 363, 369, 374, 379, 385, 390, 396, 401, 406, 412, 417, 423, 428, 433, 439, 444, 450, 455, 460, 466, 471, 477, 482, 487, 493, 498, 504, 509, 514, 520, 525, 531, 536, 541, 547, 552, 558, 563, 568, 574, 579, 585, 590, 595, 601, 606, 612, 617

T

tags (bugsy.Comment attribute), 130, 135, 140, 146, 151, 157, 162, 167, 173, 178, 184, 189, 194, 200, 205, 211, 216, 221, 227, 232, 238, 243, 248, 254, 259, 265, 270, 275, 281, 286, 292, 297, 302, 308, 313, 319, 324, 329, 335, 340, 346, 351, 356, 362, 367, 373, 378, 383, 389, 394, 400, 405, 410, 416, 421, 427, 432, 437, 443, 448, 454, 459, 464, 470, 475, 481, 486, 491, 497, 502, 508, 513, 518, 524, 529, 535, 540, 545, 551, 556, 562, 567, 572, 578, 583, 589, 594, 599, 605, 610, 616
text (bugsy.Comment attribute), 130, 135, 141, 146, 151, 157, 162, 168, 173, 178, 184, 189, 195, 200, 205, 211, 216, 222, 227, 232, 238, 243, 249, 254, 259, 265, 270, 276, 281, 286, 292, 297, 303, 308, 313, 319, 324, 330, 335, 340, 346, 351, 357, 362, 367, 373, 378, 384, 389, 394, 400, 405, 411, 416, 421, 427, 432, 438, 443, 448, 454, 459, 465, 470, 475, 481, 486, 492, 497, 502, 508, 513, 519, 524, 529, 535, 540, 546, 551, 556, 562, 567, 573, 578, 583, 589, 594, 600, 605, 610, 616
time (bugsy.Comment attribute), 130, 135, 141, 146, 151, 157, 162, 168, 173, 178, 184, 189, 195, 200, 205, 211, 216, 222, 227, 232, 238, 243, 249, 254, 259, 265, 270, 276, 281, 286, 292, 297, 303, 308, 313, 319, 324, 330, 335, 340, 346, 351, 357, 362, 367, 373, 378, 384, 389, 394, 400, 405, 411, 416, 421, 427, 432, 438, 443, 448, 454, 459, 465, 470, 475, 481, 486, 492, 497, 502, 508, 513, 519, 524, 529, 535, 540, 546, 551, 556, 562, 567, 573, 578, 583, 589, 594, 600, 605, 610, 616
timeframe() (bugsy.Search method), 131, 137, 142, 147, 153, 158, 164, 169, 174, 180, 185, 191, 196, 201, 207, 212, 218, 223, 228, 234, 239, 245, 250, 255, 261, 266, 272, 277, 282, 288, 293, 299, 304, 309, 315, 320, 326, 331, 336, 342, 347, 353, 358, 363, 369, 374, 380, 385, 390, 396, 401, 407, 412, 417, 423, 428, 434, 439, 444, 450, 455, 461, 466, 471, 477, 482, 488, 493, 498, 504, 509, 515, 520, 525, 531, 536, 542, 547, 552, 558, 563, 569, 574, 579, 585, 590, 596, 601, 606, 612, 617
to_dict() (bugsy.Bug method), 129, 134, 139, 145, 150, 156, 161, 166, 172, 177, 183, 188, 193, 199, 204, 210, 215, 220, 226, 231, 237, 242, 247, 253, 258, 264, 269, 274, 280, 285, 291, 296, 301, 307, 312, 318, 323, 328, 334, 339, 345, 350, 355, 361, 366, 372, 377, 382, 388, 393, 399, 404, 409, 415, 420, 426, 431, 436, 442, 447, 453, 458, 463, 469, 474, 480, 485, 490, 496, 501, 507, 512, 517, 523, 528, 534, 539, 544, 550, 555, 561, 566, 571, 577, 582, 588, 593, 598, 604, 609, 615

U

update() (bugsy.Bug method), 129, 134, 140, 145, 150, 156, 161, 167, 172, 177, 183, 188, 194, 199, 204, 210, 215, 221, 226, 231, 237, 242, 248, 253, 258, 264, 269, 275, 280, 285, 291, 296, 302, 307, 312, 318, 323, 329, 334, 339, 345, 350, 356, 361, 366, 372, 377, 383, 388, 393, 399, 404, 410, 415, 420, 426, 431, 437, 442, 447, 453, 458, 464, 469, 474, 480, 485, 491, 496, 501, 507, 512, 518, 523, 528, 534, 539, 545, 550, 555, 561, 566, 572, 577, 582, 588, 593, 599, 604, 609, 615

V

version (bugsy.Bug attribute), 129, 134, 140, 145, 150, 156, 161, 167, 172, 177, 183, 188, 194, 199, 204, 210, 215, 221, 226, 231, 237, 242, 248, 253, 258, 264, 269, 275, 280, 285, 291, 296, 302, 307, 312, 318, 323, 329, 334, 339, 345, 350, 356, 361, 366, 372, 377, 383, 388, 393, 399, 404, 410, 415, 420, 426, 431, 437, 442, 447, 453, 458, 464, 469, 474, 480, 485, 491, 496, 501, 507, 512, 518, 523, 528, 534, 539, 545, 550, 555, 561, 566, 572, 577, 582, 588, 593, 599, 604, 609, 615

W

whiteboard() (bugsy.Search method), 131, 137, 142, 147, 153, 158, 164, 169, 174, 180, 185, 191, 196, 201, 207, 212, 218, 223, 228, 234, 239, 245, 250, 255, 261, 266, 272, 277, 282, 288, 293, 299, 304, 309, 315, 320, 326, 331, 336, 342, 347, 353, 358, 363, 369, 374, 380, 385, 390, 396, 401, 407, 412, 417, 423, 428, 434, 439, 444, 450, 455, 461, 466, 471, 477, 482, 488, 493, 498, 504, 509, 515, 520, 525, 531, 536,

542, 547, 552, 558, 563, 569, 574, 579, 585,
590, 596, 601, 606, 612, 617