Bugzilla to GitHub via Phobos — A Love Letter

Dr. Robert Schadek

August 3, 2022

`github.com/burner/D_bugzilla_to_github`

D is the best programming language!

phobos is the best standard library!

D needs more people!

Bugzilla to Github

Bugzilla to Github

- Github has 83 million users
- issue.dlang.org has maybe 12

The approach

1. Get open issues from bugzilla
2. Get issue data via rest api
3. Find issues participates on github
4. A bit of markdown formatting
5. Push to issues to github

THIS IS REALLY REALLY BORING

–

WE ARE NOT GOING TO TALK ABOUT THIS

The interesting find

- the headers of all ten files in the project look the same

source/analysis.d

```
1  import std.algorithm.iteration;
2  import std.algorithm.sorting;
3  import std.array;
4  import std.conv : to;
5  import std.datetime.systime;
6  import std.exception;
7  import std.format;
8  import std.json;
9  import std.net.curl;
10 import std.range : ElementEncodingType;
11 import std.stdio;
12 import std.string : stripRight;
13 import std.traits;
14 import std.typecons : Nullable, nullable;
```

source/app.d

```
1  import std.algorithm.iteration : filter, joiner, map, fold, uniq;
2  import std.algorithm.searching : all, canFind;
3  import std.algorithm.sorting : sort;
4  import std.array;
5  import std.ascii : isASCII;
6  import std.exception;
7  import std.file;
8  import std.format;
9  import std.json;
10 import std.range : chain, chunks, zip;
11 import std.range : iota;
12 import std.stdio;
13 import std.string : indexOf;
14 import std.traits;
15 import std.typecons;
16 import std.uni : toLower;
```

source/getopenissues.d

```
1  import std.algorithm.iteration;
2  import std.algorithm.searching;
3  import std.algorithm.sorting;
4  import std.array;
5  import std.conv;
6  import std.datetime;
7  import std.net.curl;
8  import std.range : tee;
9  import std.regex;
10 import std.stdio;
11 import std.string;
12 import std.range;
13 import std.uni : isNumber;
```

source/github.d

```
1  import std.algorithm.iteration : filter, map;
2  import std.algorithm.searching : all, startsWith;
3  import std.array;
4  import std.ascii : isASCII;
5  import std.conv : to;
6  import std.format;
7  import std.json;
8  import std.range : chain;
9  import std.stdio;
10 import std.typecons : Nullable, nullable;
11
12 import requests;
13
14 import core.thread;
15 import core.time;
```

source/graphql.d

```
1  import std.algorithm.iteration : map, splitter;
2  import std.algorithm.searching : canFind, endsWith, startsWith;
3  import std.array;
4  import std.conv;
5  import std.datetime;
6  import std.exception;
7  import std.file : readText;
8  import std.format : format;
9  import std.json;
10 import std.stdio;
11 import std.traits : Unqual, isArray;
```

source/json.d

```
1  import std.algorithm.iteration : map;
2  import std.array;
3  import std.datetime.date;
4  import std.datetime.systime;
5  import std.exception : enforce;
6  import std.format;
7  import std.json;
8  import std.range : ElementEncodingType;
9  import std.stdio;
10 import std.string : stripRight;
11 import std.traits;
12 import std.typecons;
```

source/markdown.d

```
1 import std.algorithm.iteration : filter, joiner, map, fold, uniq,
      splitter;
2 import std.algorithm.searching : canFind, endsWith, startsWith;
3 import std.algorithm.sorting : sort;
4 import std.array;
5 import std.conv : to;
6 import std.exception;
7 import std.format;
8 import std.stdio;
9 import std.typecons;
```

source/rest.d

```
1  import std.algorithm.iteration;
2  import std.algorithm.sorting;
3  import std.array;
4  import std.conv : to;
5  import std.datetime.date;
6  import std.datetime.systime;
7  import std.exception;
8  import std.format;
9  import std.json;
10 import std.net.curl;
11 import std.range : ElementEncodingType, chunks;
12 import std.stdio;
13 import std.string : stripRight;
14 import std.traits;
15 import std.typecons : Nullable, nullable;
```

The work

Parsing (the good)

```
1 JSONValue getBug(long id) {
2   string url = "https://issues.dlang.org/rest/bug/%d";
3   string withId = format(url, id);
4   auto content = getContent(withId).to!string().parseJSON();
5   return content;
6 }
```

Parsing (the good)

```d
JSONValue getBug(long id) {
  string url = "https://issues.dlang.org/rest/bug/%d";
  string withId = format(url, id);
  auto content = getContent(withId).to!string().parseJSON();
  return content;
}
```


```
{"bugs":[{"priority":"P1","assigned_to_detail":{"email":"nobody","real_name":"No Owner","name":"nobody",
    "id":606},"blocks":[],"creator":"black80","last_change_time":"2019-08-04T15:29:18Z","
    is_cc_accessible":true,"keywords":[],"creator_detail":{"email":"black80","real_name":"a11e99z","
    name":"black80","id":2335},"cc":["r.sagitario"],"url":"","assigned_to":"nobody","groups":[],"
    see_also":[],"id":20005,"whiteboard":"","creation_time":"2019-06-25T14:10:08Z","qa_contact":"","
    depends_on":[],"dupe_of":null,"resolution":"FIXED","classification":"Unclassified","alias":[],"
    op_sys":"Windows","status":"RESOLVED","cc_detail":[{"email":"r.sagitario","real_name":"Rainer
    Schuetze","name":"r.sagitario","id":648}],"summary":"VC++ can exists in separate BuildTools folder
     (not only in Community\\Enterprise)","is_open":false,"platform":"x86_64","severity":"enhancement"
    ,"flags":[],"version":"D2","deadline":null,"component":"visuald","is_creator_accessible":true,"
    product":"D","is_confirmed":true,"target_milestone":"---"}],"faults":[]}
```

Parsing (the good)

```
1  struct Bug {
2    long id;
3    Nullable!SysTime actual_time;
4    Nullable!(long[]) alias_;
5    string assigned_to;
6    Person assigned_to_detail;
7    long[] blocks;
8    string[] cc;
9    Person[] cc_detail;
10   string classification;
11   string component;
12   SysTime creation_time;
13   string creator;
14   Person creator_detail;
15   Nullable!(SysTime) deadline;
16   long[] depends_on;
```

Parsing (the good)

```
1  T tFromJson(T)(JSONValue js) {
2    T ret;
3    static if(isArray!T && !isSomeString!T) {{
4      if(js.type() == JSONType.array) {
5        ret ~= js.arrayNoRef()
6          .map!(it => tFromJson!(ElementEncodingType!T)(it))
7          .array;
8      }
9    }} else static if(is(T : Nullable!Z, Z)) {{
10     if(mem in obj && obj.type != JSONType.null_) {
11       ret = tFromJson!(Z)(obj).nullable();
12     }
```

Parsing (the good)

```
1 import std.traits
2   : isArray
3   , isSomeString
4   , isFloatingPoint
5   , isIntegral
6   , FieldNameTuple;
```

Parsing (the good)

```
1  T tFromJson(T)(JSONValue js) {
2    T ret;
3    static if(isArray!T && !isSomeString!T) {{
4      if(js.type() == JSONType.array) {
5        ret ~= js.arrayNoRef()
6          .map!(it => tFromJson!(ElementEncodingType!T)(it))
7          .array;
8      }
9    }} else static if(is(T : Nullable!Z, Z)) {{
10     if(mem in obj && obj.type != JSONType.null_) {
11       ret = tFromJson!(Z)(obj).nullable();
12     }
```

Parsing (the good)

```
auto map(alias Func,S)(S[] input) {
  alias R = typeof(Func(S.init));
  R[] ret;
  foreach(s; input) {
    ret ~= Func(s);
  }
  return ret;
}
```

Parsing (the good)

```
1  T tFromJson(T)(JSONValue js) {
2    T ret;
3    static if(isArray!T && !isSomeString!T) {{
4      if(js.type() == JSONType.array) {
5        ret ~= js.arrayNoRef()
6          .map!(it => tFromJson!(ElementEncodingType!T)(it))
7          .array;
8      }
9    }} else static if(is(T : Nullable!Z, Z)) {{
10     if(mem in obj && obj.type != JSONType.null_) {
11       ret = tFromJson!(Z)(obj).nullable();
12     }
```

Parsing (the good)

```
1  T[] filter(alias Cond,T)(T[] input) {
2    T[] ret;
3    foreach(it; input) {
4      if(Cond(it)) {
5        ret ~= it;
6      }
7    }
8    return ret;
9  }
```

Parsing (the good)

```
1  T tFromJson(T)(JSONValue js) {
2    T ret;
3    static if(isArray!T && !isSomeString!T) {{
4      if(js.type() == JSONType.array) {
5        ret ~= js.arrayNoRef()
6          .map!(it => tFromJson!(ElementEncodingType!T)(it))
7          .array;
8      }
9    }} else static if(is(T : Nullable!Z, Z)) {{
10     if(mem in obj && obj.type != JSONType.null_) {
11       ret = tFromJson!(Z)(obj).nullable();
12     }
```

Parsing (the good)

```
1 import std.json;
2
```

Parsing (the good)

```
13    }} else static if(is(T == struct)) {{
14      enforce(js.type() == JSONType.object
15          , format("%s %s", js.type(), js.toPrettyString())
16        );
17      JSONValue[string] obj = js.objectNoRef();
18      static foreach(memPre; FieldNameTuple!(T)) {{
19        enum mem = memPre.stripRight("_");
20        alias MT = typeof(__traits(getMember, T, memPre));
21        static if(is(MT : Nullable!F, F)) {{
22          if(mem in obj && obj[mem].type != JSONType.null_) {
23            static if(is(F == SysTime)) {{
24              __traits(getMember, ret, memPre) = SysTime.
      fromISOExtString(obj[mem].get!string());
25            }} else static if(is(F == Date)) {{
26              __traits(getMember, ret, memPre) = Date.fromISOExtString(
      obj[mem].get!string());
```

Parsing (the good)

```
1  void enforce(bool cond, lazy string msg, string filename = __FILE__
2      , int line = __LINE__)
3  {
4    if(!cond) {
5      throw new Exception(msg, file, line);
6    }
7  }
```

Parsing (the good)

```
13    }} else static if(is(T == struct)) {{
14      enforce(js.type() == JSONType.object
15          , format("%s %s", js.type(), js.toPrettyString())
16        );
17      JSONValue[string] obj = js.objectNoRef();
18      static foreach(memPre; FieldNameTuple!(T)) {{
19        enum mem = memPre.stripRight("_");
20        alias MT = typeof(__traits(getMember, T, memPre));
21        static if(is(MT : Nullable!F, F)) {{
22          if(mem in obj && obj[mem].type != JSONType.null_) {
23            static if(is(F == SysTime)) {{
24              __traits(getMember, ret, memPre) = SysTime.
      fromISOExtString(obj[mem].get!string());
25            }} else static if(is(F == Date)) {{
26              __traits(getMember, ret, memPre) = Date.fromISOExtString(
      obj[mem].get!string());
```

Parsing (the good)

```
struct SysTime {
  static SysTime fromISOExtString(string input) {
    // do you really want to parse
    // YYYY-MM-DDTHH:MM:SS.FFFFFFFTZ
  }
}

struct Date {
  static Date fromISOExtString(string input) {
    // YYYY-MM-DD easier
  }
}
```

Parsing (the good)

```
13    }} else static if(is(T == struct)) {{
14      enforce(js.type() == JSONType.object
15          , format("%s %s", js.type(), js.toPrettyString())
16        );
17      JSONValue[string] obj = js.objectNoRef();
18      static foreach(memPre; FieldNameTuple!(T)) {{
19        enum mem = memPre.stripRight("_");
20        alias MT = typeof(__traits(getMember, T, memPre));
21        static if(is(MT : Nullable!F, F)) {{
22          if(mem in obj && obj[mem].type != JSONType.null_) {
23            static if(is(F == SysTime)) {{
24              __traits(getMember, ret, memPre) = SysTime.
      fromISOExtString(obj[mem].get!string());
25            }} else static if(is(F == Date)) {{
26              __traits(getMember, ret, memPre) = Date.fromISOExtString(
      obj[mem].get!string());
```

Parsing (the good)

```
27            }} else static if(is(F == struct)) {{
28              __traits(getMember, ret, memPre) = tFromJson!F(obj[mem]);
29            }} else static if(isArray!F && !isSomeString!F) {{
30              __traits(getMember, ret, memPre) = tFromJson!F(obj[mem]);
31            }} else {{
32              __traits(getMember, ret, memPre) = obj[mem].get!F();
33            }}
34          } else {
35            __traits(getMember, ret, memPre) = MT.init;
36          }
37        }} else static if(is(MT == SysTime)) {{
38          __traits(getMember, ret, memPre) = SysTime.fromISOExtString(
      obj[mem].get!string());
39        }} else static if(is(MT == struct)) {{
40          __traits(getMember, ret, memPre) = tFromJson!(MT)(obj[mem]);
41        }} else static if(isArray!MT && !isSomeString!MT) {{
42          __traits(getMember, ret, memPre) = tFromJson!(MT)(obj[mem]);
```

Parsing (the good)

```
43          }} else {{
44            static if(is(MT == bool)) {
45              if(obj[mem].type == JSONType.true_) {
46                __traits(getMember, ret, memPre) = true;
47              } else if(obj[mem].type == JSONType.false_) {
48                __traits(getMember, ret, memPre) = false;
49              } else {
50                __traits(getMember, ret, memPre) = obj[mem].get!long()!=0;
51              }
52            } else {
53              __traits(getMember, ret, memPre) = obj[mem].get!MT();
54            }
55          }}
56        }}
57      }}
58      return ret;
59  }
```

Parsing (the ugly)

```
1    <tr id="b22800" class="bz_bugitem bz_normal bz_P1 bz_NEW bz_row_odd">
2      <td class="first-child bz_id_column">
3        <a href="show_bug.cgi?id=22800">22800</a>
4        <span class="bz_default_hidden"></span>
5      </td>
6      <td class="bz_product_column nowrap">
7          <span title="D">D
8          </span>
9      </td>
10     <td class="bz_component_column nowrap">
11         <span title="phobos">phobos
12         </span>
13     </td>
14     <td class="bz_assigned_to_column nowrap">
15         <span title="nobody">nobody
16         </span>
17     </td>
18     <td class="bz_bug_status_column nowrap">
19         <span title="NEW">NEW
20         </span>
21     </td>
22     <td class="bz_resolution_column nowrap">
23         <span title="---">---
24         </span>
25     </td>
26     <td class="bz_short_desc_column">
27         <a href="show_bug.cgi?id=22800">DDOC throw section for writeln is incomplete        </a>
28     </td>
```

Parsing (the ugly)

```
1  public struct BugDate {
2    long id;
3    Date date;
4  }
5
6  public BugDate[] getOpenIssuesImpl(string component) {
7    string temp = `https://issues.dlang.org/buglist.cgi?component=`
8      ~ component
9      ~ `&limit=0&order=changeddate%20DESC%2Cbug_id&product=D&
       query_format=advanced&resolution=---`;
10
11   string page = () @trusted { return cast(string)get(temp); }();
12   Date[] d = splitDateTimes(page);
13   long[] i = splitIds(page);
14   assert(d.length == i.length, format("%s %s", d.length, i.length));
```

Parsing (the ugly)

```
15    return zip(i, d)
16      .map!(id => BugDate(id[0], id[1]))
17      .array
18      .uniq
19      .array;
20  }
```

Parsing (the ugly)

```
1  struct Zip(T,R) {
2    T a;
3    R b;
4  }
5
6  Zip!(T,R)[] zip(T,R)(T[] a, R[] b) {
7    enforce(a.length == b.length);
8    Zip!(T,R)[] ret;
9    foreach(idx, it; a) {
10     ret ~= Zip!(T,R)(it, b[idx]);
11   }
12   return ret;
13 }
```

Parsing (the ugly)

```
15    return zip(i, d)
16      .map!(id => BugDate(id[0], id[1]))
17      .array
18      .uniq
19      .array;
20  }
```

Parsing (the ugly)

```
1 T[] uniq(alias pred, T)(T[] input) {
2   T[] ret;
3   foreach(idx, it; input) {
4     if(idx == 0 || !pred(it, ret[$ - 1])) {
5       ret ~= it;
6     }
7   }
8   return ret;
9 }
```

Parsing (the ugly)

```
21 long[] splitIds(string page) {
22   enum re = ctRegex!(`"show_bug.cgi\?id=[0-9]+"`);
23   auto m = page.matchAll(re);
24
25   return m
26     .filter!(it => it.length > 0)
27     .map!(it => it.front)
28     .map!(it => it.find!(isNumber))
29     .map!(it => it.until!(it => !it.isNumber()))
30     .filter!(it => !it.empty)
31     .map!(it => it.to!long())
32     .uniq
33     .array;
34 }
```

Parsing (the ugly)

```
1 T[] find(alias pred, T)(T[] input) {
2   while(!input.empty && !pred(input.front)) {
3     input.popFront();
4   }
5   return input;
6 }
```

Parsing (the ugly)

```
21 long[] splitIds(string page) {
22   enum re = ctRegex!(`"show_bug.cgi\?id=[0-9]+"`);
23   auto m = page.matchAll(re);
24
25   return m
26     .filter!(it => it.length > 0)
27     .map!(it => it.front)
28     .map!(it => it.find!(isNumber))
29     .map!(it => it.until!(it => !it.isNumber()))
30     .filter!(it => !it.empty)
31     .map!(it => it.to!long())
32     .uniq
33     .array;
34 }
```

Parsing (the ugly)

```
1 T[] until(alias pred, T)(T[] input) {
2   T[] ret;
3   while(!input.empty && !pred(input.front)) {
4     ret ~= input.front;
5   }
6   return ret;
7 }
```

Parsing (the ugly)

```
21 long[] splitIds(string page) {
22   enum re = ctRegex!(`"show_bug.cgi\?id=[0-9]+"`);
23   auto m = page.matchAll(re);
24
25   return m
26     .filter!(it => it.length > 0)
27     .map!(it => it.front)
28     .map!(it => it.find!(isNumber))
29     .map!(it => it.until!(it => !it.isNumber()))
30     .filter!(it => !it.empty)
31     .map!(it => it.to!long())
32     .uniq
33     .array;
34 }
```

Parsing (the ugly)

```
1 T to(T,S)(S input) {
2   // best function ever
3   // \U1F4A9 in \u2666 out
4 }
```

Parsing (the ugly)

At this point I was thinking to redo splitIds without phobos, then I saw I missed to talk about `std.regex`.

splitIds

```
long[] splitIds(string page) {
  enum re = ctRegex!(`"show_bug.cgi\?id=[0-9]+"`);
  auto m = page.matchAll(re);

  long[] ret;
  foreach(it; m) {
    if(it.empty) {
      continue;
    }
    while(!it.empty && !(it.front >= '0' && it.front <= '9')) {
      it.popFront();
    }
    if(it.empty) {
      continue;
    }

    long num;
    long mul = 1;
    while(!it.empty && it.front >= '0' && it.front <= '9') {
      long t = (cast(char)it.front) - '0';
      num *= mul;
      num += t;
      mul *= 10;
      it.popFront();
    }

    if(mul == 1) {
      continue;
```

Finishing the phobos Tutorial

std.format

```
1  assert(format("%s", Hello) == "Hello");
2  assert(format("%5.f", 1.3333333) == "1.33333");
3  assert(format("%2$s %1$s", "a", "b") == "b a");
4  assert(format("%,3d", 100000000) == "100,000,000");
5  assert(format("%(%s,%)", [1,2,3]) == "1,2,3");
6
7  struct Foo {
8    int a;
9  }
10 assert(format("%s", Foo.init) == "Foo(0)");
```

std.algorithm.searching

- `find`
  - `canFind`
  - `until`
  - `countUntil`
- `startsWith`
- `endsWith`

Not everything is Roses and Rainbows

Common Complaints

- GC
- Exceptions
- too much coupling
- auto decoding

Common Complaints

- GC
  - Humankind is not smart enough for manual memory management
- Exceptions
- too much coupling
- auto decoding

Common Complaints

- GC
  - Humankind is not smart enough for manual memory management
- Exceptions
  - Unexpected things happen `SysTime SysTime.fromISOExtString(string);`
  - `Nullable!SysTime` is 💩
  - `Result!(SysTime,Error)` is 💩
  - `errno` is 💩
  - did some say `throw new Exception("Invalid Timezone")`
- too much coupling

- auto decoding

Common Complaints

- GC
  - Humankind is not smart enough for manual memory management
- Exceptions
  - Unexpected things happen `SysTime SysTime.fromISOExtString(string);`
  - `Nullable!SysTime` is 💩
  - `Result!(SysTime,Error)` is 💩
  - `errno` is 💩
  - did some say `throw new Exception("Invalid Timezone")`
- too much coupling
  - no global state shared
  - no reason to re-implement `find` over and over again
  - most of phobos functions are `pure`
  - this argument is just a red herring
- auto decoding

Common Complaints

- GC
  - Humankind is not smart enough for manual memory management
- Exceptions
  - Unexpected things happen `SysTime SysTime.fromISOExtString(string);`
  - `Nullable!SysTime` is 💩
  - `Result!(SysTime,Error)` is 💩
  - `errno` is 💩
  - did some say `throw new Exception("Invalid Timezone")`
- too much coupling
  - no global state shared
  - no reason to re-implement `find` over and over again
  - most of phobos functions are `pure`
  - this argument is just a red herring
- auto decoding
  - decision between default incorrectness and scapegoating

Too few things

- no html
- no xml
- no yaml
- no SI units
- no eventcore
- no dmd frontend (I want CT D parsing)
- too little coupling
- why is numpy/keras/tensorflow not in phobos
  - CT parsing of python type annotations

Back to the Topic

Back to the Topic

- To get more things, we need more people

Back to the Topic

- To get more things, we need more people
- We can not tell people

Back to the Topic

- To get more things, we need more people
- We can not tell people

- But we can ask people to contribute

Back to the Topic

- To get more things, we need more people
- We can not tell people

- But we can ask people to contribute

- Nobody hears us on bugzilla, on github they might

TL;DR

phobos everywhere

bugzilla $\rightarrow$ github

Questions?