Improving App Development in Vala

Princeton Ferro

June 22, 2021

Outline

A language server for Vala

A language server for Vala

A language server is the first big step to improving the Vala developer experience.

- Language servers provide code intelligence.
- Code intelligence helps you iterate faster.
- Iterating faster means you develop higher-quality apps.

Precursors

- Valama IDE
- vala-pack (GNOME Builder)
- gedit-code-assistance
- Anjuta

These weren't very good.

If you wanted Vala to be supported on *N* editors, you had to write *N* plugins.

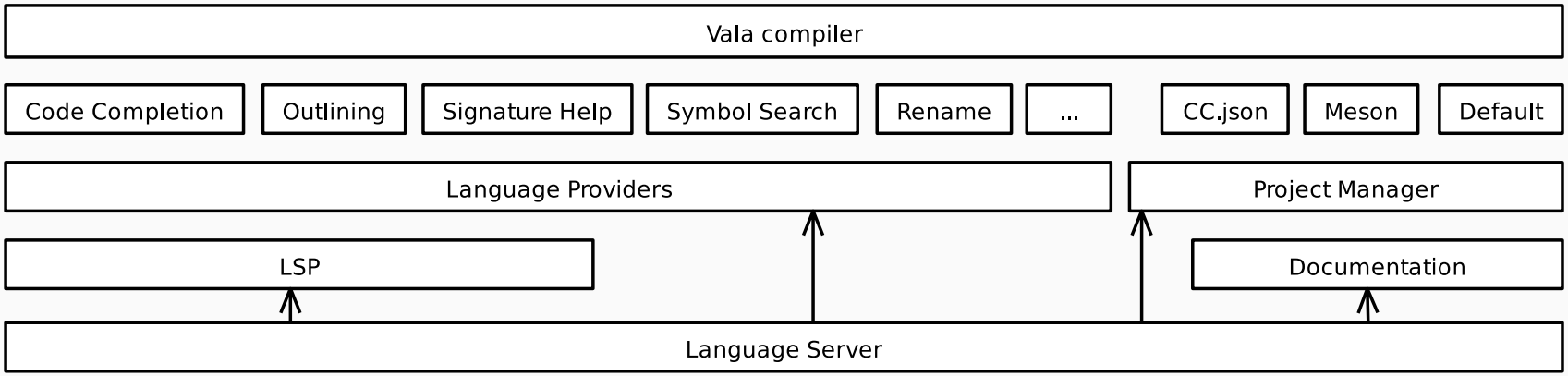
The Language Server Protocol

Allows you to write *one* language server for *N* editors.

Defines how an editor should ask a language server for diagnostics, completion suggestions, etc.

Very flexible–during handshake, server and client advertise what methods they support.

Architecture



Architecture of VLS

Architecture

Initial problems:

- Plugging compiler memory leaks
  - Reduced average memory consumption from a few GB to a few hundred MB
- Recovering from syntax errors

Architecture

User interaction must be fast:

- Delay context updates while user is typing
- Use backwards parser to extract simple expressions for rapid completion

Architecture

The server–not the plugin–should know as much about the build system as possible, since we don't want to reimplement project management across *N* editors.

Architecture

- Integrate with Meson build system
  - Has a friendly introspection API
  - Vala is a first-class citizen
  - Used by >95% of Vala projects
- Autotools introspection is too complex, not worth the effort
- CMake introspection API is okay but Vala targets aren't recognized
  - We'd first need to write a CMake plugin for Vala (and convince people to use it)

Architecture

Navigation:

- Show references
- Go to implementation(s)
- Go to base/hidden method

Code refactoring:

- Rename symbol

Architecture

For documentation, we read the GObject Introspection files–which usually come installed with the library–and parse them according to the GTK-Doc[1] and ValaDoc[2] markup languages.

The GTK-Doc comments are usually written for the C version of the API. Therefore, in a second step, we map the C identifiers to Vala identifiers. This produces documentation that more closely resembles Vala code.

[1] https://developer.gnome.org/gtk-doc-manual/stable/
[2] https://valadoc.org/markup.htm

Editor support

Today you can get code intelligence for Vala in a lot of editors, thanks to plugins and built-in support.

- Visual Studio Code
- GNOME Builder
- Neovim and vim8
- Kate
- Emacs
- Sublime Text

Editor support

Visual Studio Code

- Install Vala plugin

GNOME Builder

- Bundled with VLS
- Enable ”Vala Language Server” and disable ”GVLS”

Editor support

vim8/neovim

- Install coc.nvim
- Add this to your config (`:CocConfig`)
- Works well with vista.vim plugin.

```
"languageserver": {
    "vala": {
        "command": "vala-language-server",
        "filetypes": ["vala", "genie"]
    }
}
```

Or you can install nvim-lspconfig for neovim.

Editor support

Kate

- Enable built-in LSP plugin

Emacs

- Install lsp-mode

Editor support

Sublime Text

- Install the Vala-TMBundle and LSP packages
- Add config below to `LSP.sublime-settings`
- `Tools > LSP > Enable Language Server Globally... > vala-language-server`

```
"clients": {
    "vala-language-server": {
        "command": [
            "/usr/bin/vala-language-server"
        ],
        "selector": "source.vala | source.genie"
    },
}
```

Future plans

What's next?

Responsiveness

For large projects (> 20000 lines), VLS is slow.

Off-main-thread compilation or incremental recompilation?

- Threaded compilation: fast and ”simple”, but uses much more memory (code context duplication)
- Incremental recompilation: less memory (reuse existing code context), faster, but is much more complex to implement
  - Relies on the observation that most of the code context is unchanged as you type
  - Corner case–refactoring commonly-used symbols

Code refactoring

Things on my radar:

- Organize imports
- Extract to variable/method/...
- Inline
- Change signature of a method
- Delete method

Completion

There are a *lot* of additional cases to be covered. We have multiple issues tracking this.[3]

In general, we want code suggestions to be context-sensitive, and to work without configuration.

[3] https://github.com/Prince781/vala-language-server/milestone/2

Smart method completion

For example, we want to complete a method. Do we do this:

```
method (...args)
```

or this:

```
method(...args)
```

Most projects use the first style, but a few notable ones (Geary) use the second.

Rather than expose a configuration option, we should understand the coding style and always do the right thing.

Global completion

Show symbols within namespaces we haven't imported, and import the appropriate namespace for the selected symbol.

Improving Vala

Improving Vala

A year and a half ago, none of this was possible. Writing Vala was a bare experience.

Things are better now, but there's still much room for improvement.

Vala still needs better tooling and better infrastructure.

This is key to improving the Vala ecosystem.

Improving Vala

Lets examine other areas:

- Tooling
  - Static analysis
  - Linting and Formatting
  - Templating
  - Dependency management
- Website
- Documentation
- Community

Static analysis

It would be nice to have something like Clang's `scan-build` or GCC's `-fanalyzer` for Vala.

These tell you about deeper errors in your code and usually don't care about showing you some false positives. (For example, null pointer access or use-after-free.)

For Vala we could have reference cycle detection.

Static analysis

Reference cycle detection could give warnings when using cyclic data structures improperly.

Bad:

```
class List<T> {
    public T data;
    public List<T>? prev;
    public List<T>? next;

    public List<T> add (T data) {
        next = new List<T> () {
            data = data,
            prev = this
        };
        return next;
    }
}
```

Good:

```
class List<T> {
    public T data;
    public weak List<T>? prev;
    public List<T>? next;

    public List<T> add (T data) {
        next = new List<T> () {
            data = data,
            prev = this
        };
        return next;
    }
}
```

Static analysis

Or when creating circular references in subtle ways...

```
class Zombie {
    public bool sort_ascend;
    public int age { get; private set; }
    public Gee.Set<Zombie> children { get; private set; }

    public Zombie (int age) {
        this.age = age;
        children = new Gee.TreeSet<Zombie> (
            (a, b) => sort_ascend ? a.age - b.age : b.age - a.age
        );
    }
}
```

Closure holds strong reference to `this`.

Static analysis

Or when creating circular references in subtle ways...

```
class Zombie {
    public bool sort_ascend;
    public int age { get; private set; }
    public Gee.Set<Zombie> children { get; private set; }
    static CompareDataFunc<Zombie> make_compare (Zombie _self) {
        weak Zombie self = _self;
        return (a, b) =>
            self.sort_ascend ? a.age - b.age : b.age - a.age;
    }
    public Zombie (int age) {
        this.age = age;
        children = new Gee.TreeSet<Zombie> (make_compare (this));
    }
}
```

Closure now holds a weak reference to `this`.

Linting and Formatting

Linters suggest fixes to code structure and semantics. They catch errors and unintended behavior.

Formatters are basically code ”prettifiers.”

Vala has `vala-lint`, which is mostly a code prettifier analogous to `rustfmt`.

We also need something like `rust-clippy`.

Linting

For example, `rust-clippy` has hundreds of lints, can catch common programming mistakes, and offers suggestions for fixing them.

A sample:

```
fn main() {
    let a = 1.231f32;
    let b = 1.232f32;

    if a == b {
        print!("a == b");
    } else {
        print!("a != b");
    }
}
```

Linting

```
% clippy-driver comparef32.rs
error: strict comparison of `f32` or `f64`
 --> comparef32.rs:5:8
  |
5 |     if a == b {
  |        ^^^^^^ help: consider comparing them within some
  ↪  margin of error: `(a - b).abs() < error_margin`
  |
  = note: `#[deny(clippy::float_cmp)]` on by default
  = note: `f32::EPSILON` and `f64::EPSILON` are available for
  ↪  the `error_margin`
  = help: for further information visit
https://rust-lang.github.io/rust-clippy/master/index.html#float_cmp

error: aborting due to previous error
```

Templating

Templates allow developers to zoom past project setup and jump straight to writing code.

For C♯, there's officially `dotnet new`, which allows you to initialize a new project from a collection of community-made templates.

Other languages have similar unofficial tools.

Templating

Recently I introduced `valdo`[4] with the same idea.

```
% valdo
Available templates:
--------------------
new - a bare app, with minimal dependencies
lib - a bare library with minimal dependencies
gtk - a starter GTK3 app
```

I think it would be good if this takes off.

I encourage anyone with ideas to submit PRs for new templates.

[4] https://github.com/Prince781/valdo

Dependency management

Rust/JS/Go/C♯ dependencies are per-project, statically linked in or bundled.

Vala dependencies are system-wide, dynamically linked in.

How to bridge the gap?

Website

GNOME Projects / Vala Home RecentChanges Schedule PrincetonFerro Settings Logout

Vala - Compiler Using the GObject Type System

Introduction

Vala is a programming language using modern high level abstractions without imposing additional runtime requirements and without using a different ABI compared to applications and libraries written in C. Vala uses the GObject type system and has additional code generation routines that make targeting the GNOME stack simple. Vala has many other uses where native binaries are required. More...

Contents

News

- June 4, 2021: Vala 0.52.4 released [src] (Stable)
- June 4, 2021: Vala 0.50.9 released [src] (Stable)
- June 9, 2021: Vala 0.48.18 released [src] (Long-term Support)
- January 11, 2021: Vala 0.40.25 released [src] (Long-term Support)
- August 10, 2020: Vala 0.46.13 released [src] (Stable) [EOL]
- August 14, 2019: Vala 0.36.20 released [src] (Long-term Support) [EOL]

The current stable release branch with Long-Term Support is 0.48.

The current website[5] isn't very friendly to newcomers

[5] https://wiki.gnome.org/Projects/Vala

Website

Vala

About Learn Community API Reference Source Code News

A Modern, Fast, Open Source Language

Be able to write modern, high-level code, with fully native performance, without requiring any additional runtime and maintaining full API/ABI compatibility with your C applications and libraries.

GET STARTED WHY VALA?

```
int main (string[] args) {
  var app = new Gtk.Application (
    "io.github.myteam.MyApp",
    ApplicationFlags.FLAGS_NONE
  );

  app.startup.connect (() => new Gtk.ApplicationWindow (app) {
    default_width = 800,
    default_height = 600,
    title = "Hello, World!"
  });

  app.activate.connect (() => app.active_window.present ());

  return app.run (args);
}
```

Familiar to anyone who has worked with Java or C#, but maintaining API/ABI compatibility with C.

Low memory requirements, fully native execution, and created targeting the GLib object system.

Signals, properties, generics, lambdas, assisted memory management, exception handling, type inference, asynchronous programming, and more.

Proposal for a new website[6] at vala-lang.org

[6] https://github.com/nahuelwexd/vala-website

Documentation

valadoc.org is very good, but it could be better

It's mostly an API browser with some code snippets.

There are existing tutorials out there. Can we centralize everything (APIs and tutorials) into one website?

- https://naaando.gitbooks.io/the-vala-tutorial/content/en/1-introduction/what-is-vala.html
- https://wiki.gnome.org/Projects/Vala/Tutorial
- https://developer.gnome.org/gnome-devel-demos/stable/beginner.vala.html

Community

`#vala`[7] is nice, but IRC isn't for everyone.

Recently a Vala Discord[8] has been created. It's become quite popular. I encourage people to join!

Also a Twitter account[9] has been created to promote the language. Consider following it.

Thanks for listening!

[7] irc://irc.gnome.org/vala
[8] https://discord.gg/YFAzjSVHt7
[9] https://twitter.com/vala_lang