The GNOME Conference
GUADEC

Showing Up for Python in GNOME

Dan Yeaw (dan@yeaw.me)

About Me

- Dan Yeaw (pronounced: Yaw)
- Originally from California, now lives in Michigan
- Co-maintainer of Gaphor, SysML/UML Modeling tool (GNOME Circle)
- Co-maintainer of Gvsbuild for building GTK on Windows
- GNOME Foundation member, developer access this year
- Hosts Michigan Python monthly
- Works for Ford Motor Company on Functional Safety

About Me
- Dan Yeaw (pronounced: Yaw)
- Originally from California, now lives in Michigan
- Co-maintainer of Gaphor, SysML/UML Modeling tool (GNOME Circle)
- Co-maintainer of Gvsbuild for building GTK on Windows
- GNOME Foundation member, developer access this year
- Hosts Michigan Python monthly
- Works for Ford Motor Company on Functional Safety

Hi, I'm Dan Yeaw, and I'm sooo excited to talk to you about Showing up for Python in GNOME!!

Unleashing Interests with Python

```
>>> import pypokedex
>>> pokemon = pypokedex.get(name="Decidueye")
>>> pokemon.name
'decidueye'
>>> pokemon.types
['grass', 'ghost']
>>> pokemon.base_stats
BaseStats(hp=78, attack=107, defense=75, sp_atk=100, sp_def=100)
```

Unleashing Interests with Python

```
>>> import pypokedex
>>> pokemon = pypokedex.get(name="Decidueye")
>>> pokemon.name
'decidueye'
>>> pokemon.types
['grass', 'ghost']
>>> pokemon.base_stats
BaseStats(hp=78, attack=107, defense=75, sp_atk=100, sp_def=100)
```

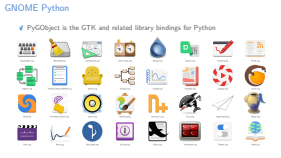
With a little help, my 10 year old son can figure out how make simple games and apps with Python, his latest project is trying to make a Pokedex for Pokemon. He loves geeking out on Pokemon, and that it is so much fun to see people get deep in to their interests!

Python is easy to learn, but hard to master - it scales easily with your skillset. It is such an important language for our ecosystem. It is often used by students, researchers, and non-professional programmers. It is used by artists, data scientists, web developers, sysadmins, and astronomers.

With Builder, Workbench, and Flatpak, it has never been easier to build an app for GNOME. This is the big tent, that we need, where we make room for and make it easy for all kinds of people to build small apps for their projects to geek out on their interests. Python is the perfect language this - so we need to make sure the GNOME experience with it lives up to this.

GNOME Python

- PyGObject is the GTK and related library bindings for Python



PyGObject is Python in GNOME. It is the successor to PyGTK that James Henstridge started in 1998 that uses gobject-introspection directly to allow you to build GNOME apps using Python. If you see the patterns in the app icons here, those deep interests that people geek out on includes drawing and art, modeling, graphing, music, genealogy, manga, classic gaming, and scientific reports.

On PyGObject

> *The current state of the Python bindings for GObject-based libraries is making it really hard to recommend using Python as a language for developing GTK and GNOME applications.*
>
> Emmanuele Bassi (2022)

On PyGObject

The current state of the Python bindings for GObject-based libraries is making it really hard to recommend using Python as a language for developing GTK and GNOME applications.

Emmanuele Bassi (2022)

In December 2022, Emmanuele Bassi wrote a blog post called "On PyGObject" with a call to action to get involved to help Christoph Reiter, the maintainer of the library. He went through 3 large use cases of features missing in PyGObject which was really make it hard to recommend. These included:

1. Typed Instances for things like GtkExpression, GtkRenderNode, and GtkEvent. These Foundational Types were supported since they aren't based on GObjects.
2. The base wrapper for GObject itself is written in Python instead of using gobject-introspection directly. However, this means that constructing and disposing of objects can be difficult because PyGObject doesn't automatically get access to the functions for doing those operations.
3. Documentation is spread out everywhere, and a lot of it wasn't updated for GTK4.

Commits Over Time

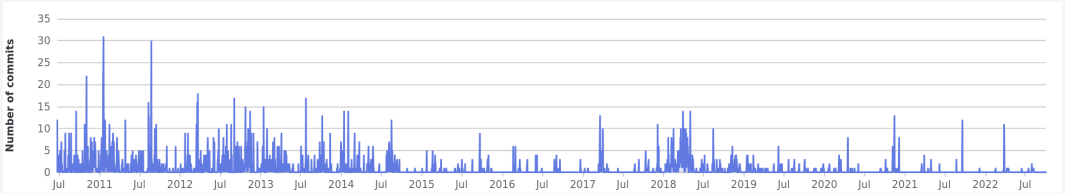
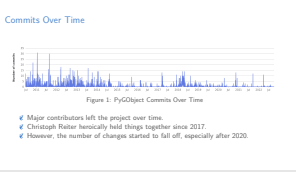


Figure 1: PyGObject Commits Over Time

- Major contributors left the project over time.
- Christoph Reiter heroically held things together since 2017.
- However, the number of changes started to fall off, especially after 2020.



You can see the pattern of this in the commit history, although it isn't the only or even most important indicator of community health. Major contributors like Simon Feltman, John Palmieri, and Martin Pitt left the project - which is a natural part of open source. Christoph kept the fires burning late in to the nights, but he was also working on other important open source projects like MSYS2. PyGObject was mostly idling along with absolutely necessary changes only. This is also the same time frame that GTK4 was released, so there was a lot of changes going on in the wider ecosystem of libraries.

Getting Involved in an Undermaintained Project

- Contributing to an undermaintained project can be difficult
- Each extra contribution is placing a burden on the developer
- Timely feedback to contributions is often not possible
- To outsiders, the GNOME project can feel hard to join, especially in these undermaintained areas

Getting Involved in an Undermaintained Project
- Contributing to an undermaintained project can be difficult
- Each extra contribution is placing a burden on the developer
- Timely feedback to contributions is often not possible
- To outsiders, the GNOME project can feel hard to join, especially in these undermaintained areas

Major open source projects really need a contribution funnel to get more people involved. This type of community building requires many hands to help newcomers, triage issues, review contributions, answer questions, and provide support.

Unfortunately, it is all too common to have one person trying to hold everything together on multiple projects. However, this is a catch-22, that one person is barely holding things together, and each issue raised, each merge request submitted is extra work and burden for them.

It doesn't feel welcoming to new people if they show up to help and their contributions dead rot, it is natural to move along and spend your time on things that you feel are making a difference.

Since GNOME is a whole project and developer access is across the project, this adds an extra layer of complexity to try to figure out how to get through. For an undermaintained project, it can feel like the GNOME community is behind a city wall, and your knocking on the unmanned gate trying join.

Community Building

- The GNOME Project Handbook greatly improves clarity on how to get involved
- The GNOME Foundation could also take a greater role

Community Building
The GNOME Project Handbook greatly improves clarity on how to get involved
The GNOME Foundation could also take a greater role

Wow! The GNOME Project Handbook which was released at the end of January. is such a special resource to document for everyone how to get involved and the expectations. A big shout out to the team whole helped make that happen!

Since GNOME as a project is made up of volunteers and individuals paid by companies with their own priorities, it can often be difficult to shift resources to help out a part of the ecosystem. Emmanuele shouldn't have to write blog posts asking for people to help get involved. There may be an opportunity for the GNOME Foundation here to track the health of key GNOME projects using metrics and then provide community building support for those that are starting to have challenges to help them out before it becomes an issue.

The State of Python in GNOME

The State of Python in GNOME

Let's switch gears a bit, over the last year, PyGObject has made some significant improvements!

Issue and Merge Request Triage

- Closed about 200 issues
- Total issue count went from over 300 to 175
- Open or draft merge requests went from 30 to 19

Issue and Merge Request Triage
- Closed about 200 issues
- Total issue count went from over 300 to 175
- Open or draft merge requests went from 30 to 19

Let's start with the basics! A clean issue and merge request backlog is important for a thriving community. We made some major inroads over the last year to reduce the total open issue and merge request counts.

Although we haven't quite got all the way to just the subset that we are really going to work on, issues in this 100-200 range feels good - something that contributors can get their head around. Much more than this often feels overwhelming.

Fundamental Types

```
def on_pressed(ctrl, n_press, x, y):
    print(ctrl.get_current_event())


def window():
    ctrl = Gtk.GestureClick()
    ctrl.connect("pressed", on_pressed)
    win = Gtk.Window.new()
    win.add_controller(ctrl)
    win.show()
    return win
```


Fundamental Types

```
def on_pressed(ctrl, n_press, x, y):
    print(ctrl.get_current_event())


def window():
    ctrl = Gtk.GestureClick()
    ctrl.connect("pressed", on_pressed)
    win = Gtk.Window.new()
    win.add_controller(ctrl)
    win.show()
    return win
```

This fixes use case number 1 from the On PyGObject blog post. Now Python developers can finally use instances of fundamental types, which was one of the big blockers for people implementing custom widgets with GTK4. This original work was starting in 2010, and Arjan Molenaar brushed it off and implemented it this year.

This fixes a ton of low level issues. you'll be able to do advanced custom drawing using render nodes, as well as accessing low level windowing system event objects, in your Python applications.

https://pygobject.gnome.org

We use to have the pygobject docs hosted on read the docs. Rafael Mardojai also had a really nice PyGObject-Guide which was a tutorial based on the Python GTK+3 Tutorial by Sebastian Pölsterl. We worked with the communities to convert the projects from the GNU Free Documentation License to the LGPL, merged the tutorials with the other docs, and moved them to a more official pygobject.gnome.org subdomain.

meson-python and PDM

```
meson setup _build
meson test -C _build
```

or

```
pdm install
pdm run pytest
```


meson-python and PDM

meson setup _build
meson test -C _build

or

pdm install
pdm run pytest

We moved from the legacy setup.py to the more modern pyproject.toml. We are using Meson for the build backend and using PDM to manage the project dependencies and virtualenvs.

Modernize API Docs

- Modernize building docs using GI-DocGen and Sphinx

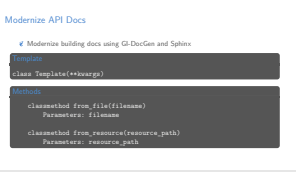
Template

```
class Template(**kwargs)
```

Methods

```
    classmethod from_file(filename)
        Parameters: filename

    classmethod from_resource(resource_path)
        Parameters: resource_path
```

Just like many other libraries have been upgrading from GTK-Doc to GI-DocGen, PyGObject also recently made the switch. GI-Docgen reuses the introspection data generated by GObject-based libraries to generate the API reference of these libraries.
Previously, we were using pgi-docgen, which was a more custom way to read GIR docs and then create a Sphinx website from them.

Previously missing documentation, like for Gtk.Template is now available and because we are using the introspection data directly less maintenance is required going forward.

Main Branch

- Small change to rename the primary branch to main
- Improves exclusivity and standardization with other GNOME projects

Main Branch
- Small change to rename the primary branch to main
- Improves exclusivity and standardization with other GNOME projects

Experimental: Asyncio Integration

- Implements Python asyncio await for Gio async results

```
async def idle_test():
    bus = await Gio.bus_get(Gio.BusType.SYSTEM)
    # Actual bus call requires more paramters
    await bus.call("org.freedesktop.NetworkManager")

policy = GLibEventLoopPolicy()
asyncio.set_event_loop_policy(policy)
loop = policy.get_event_loop()
loop.run_until_complete(idle_test())
```


Experimental: Asyncio Integration
Implements Python asyncio await for Gio async results
async def idle_test():
bus = await Gio.bus_get(Gio.BusType.SYSTEM)
# Actual bus call requires more paramters
await bus.call("org.freedesktop.NetworkManager")
policy = GLibEventLoopPolicy()
asyncio.set_event_loop_policy(policy)
loop = policy.get_event_loop()
loop.run_until_complete(idle_test())

The Future

The Future

Wheels for Windows

- Python 3.8+ no longer loads DLLs on the path
- Building GTK using MSVC with `pip install pygobject` doesn't work for getting started
- Solution: build wheels of PyGObject with the DLLs included

Wheels for Windows
- Python 3.8+ no longer loads DLLs on the path
- Building GTK using MSVC with pip install pygobject doesn't work for getting started
- Solution: build wheels of PyGObject with the DLLs included

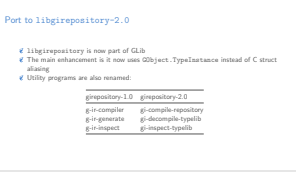
For security reasons, Python 3.8 stopped automatically loading DLLs on the path on Windows. Many libraries including PyGObject previously depended on this behavior. If you do build GTK on Windows using Gvsbuild or with MSVC directly, you don't end up with a working PyGObject without manually loading the DLLs or patching PyGObject.

We have discussed options to fix this, and there hasn't been much excitement in adding a DLL search routine in PyGObjects startup code. However, a Wheel format allows for DLLs to be bundled along side of the project though and then they are automatically loaded. This would also significantly improve install time as well, since users can directly install a pre-compiled version of PyGObject instead of compiling it during the installation.

Port to `libgirepository-2.0`

- `libgirepository` is now part of GLib
- The main enhancement is it now uses `GObject.TypeInstance` instead of C struct aliasing
- Utility programs are also renamed:

girepository-1.0	girepository-2.0
g-ir-compiler	gi-compile-repository
g-ir-generate	gi-decompile-typelib
g-ir-inspect	gi-inspect-typelib



This one is more of a chore to make sure that PyGObject is using the latest libraries. libgirepository was originally part of gobject-introspection, however it is now very stable and has been integrated with GLib to improve the build process to prevent circular dependencies between GLib and gobject-introspection.

The main change between the two versions of libgirepository is that it now uses GObject.TypeInstance as the basis of its type system, rather than simple C struct aliasing.

The symbol prefix was also updated from `g_` to `gi_`, various function arguments changed, and there were some modification to stack allocation.

Philip Withnall started this work to port PyGObject, and Arjan Molenaar has picked it up to try to bring it home.

Move API Docs

- Combine and merge the API docs to https://pygobject.gnome.org
- Would finish centralizing all docs

Move API Docs
- Combine and merge the API docs to https://pygobject.gnome.org
- Would finish centralizing all docs

Call to Action

- Contributions of any kind will help continue to help the community thrive
- Submit and help triage issues
- Continue to help us improve the docs
- Help us fix bugs and implement features
- Add examples to Workbench
- Build projects with PyGObject

Call to Action
- Contributions of any kind will help continue to help the community thrive
- Submit and help triage issues
- Continue to help us improve the docs
- Help us fix bugs and implement features
- Add examples to Workbench
- Build projects with PyGObject

Many of you have even more ideas on what we could improve next, and we would love to have your contributions!