Scripting your C++ application with Python

C++ User Group Aachen, Daniel Evers, 2017-01-12

Outline

1. What?
2. Why?
3. How?
4. Demo!
5. Links

WHAT?

What this talk is about

- Adding scripting capabilities to our C++ application
- Specifically: Adding ***Python*** scripting capabilities
  - This is “more” than adding C/C++ modules to Python
    (We already had a talk on this...)
  - But not much ;-)
- Why Python? Because ***I*** love it!
  - Your mileage may vary…
  - Feel free to use whatever you prefer.

What this talk is not...

- Extensive
  - It’s meant to be short and give you hints.
  - There’s a great CppCon 2016 talk - check the links at the end!
- Error-free
  - We’re all humans, right?

WHY?

Why integrate scripting?

- Shorten development time
  - Skip recompile cycles - just edit the script & re-run
  - Use existing Python modules
  - Get more functionality with less code in less time
  - Quickly try things before implementing them in C++
- Allow to customize your software
  - End users or project managers or integrators or ...
  - Easy to change/add business logic
  - May be easier than adding 100 parameters
  - Popular for game development (esp. AI)

How?

Basic integration steps

1. Add the Python interpreter to your app
    a. using the Python C API
2. Expose relevant APIs to Python
    a. using the Python C API

       OR
    b. using Boost.Python

       OR
    c. using Pybind11
3. Write & run scripts

Basic integration steps

1. Add the Python interpreter to your app ← this is easy
   a. using the Python C API
2. Expose relevant APIs to Python ← this can be work!
   a. using the Python C API
      OR
   b. using Boost.Python
      OR
   c. using Pybind11
3. Write & run scripts ← this depends...

Python C API

- Pros:
  - “lowest level” - all other libs are based on this
  - official API
  - provides most control
  - provides best performance
- Cons:
  - harder to use
  - more code to write
  - easier to make mistakes

Boost.Python vs. Pybind11

- both:
  - Object-oriented, C++, similar feature set
  - conversion between STL and Python types (strings, lists, …)
- Boost.Python:
  - conversions need to be manually specified
  - compiled library
- Pybind11:
  - “automagic” conversions
  - Header-only
  - requires C++11 compiler

Boost.Python vs. Pybind11

- both:
  - Object-oriented, C++, similar feature set
  - conversion between STL and Python types (strings, lists, …)
- Boost.Python: ← We had a talk on Boost.Python already
  - conversions need to be manually specified
  - compiled library
- Pybind11: ← Less work, used in the example program
  - “automagic” conversions
  - Header-only
  - requires C++11 compiler

Demo!

Links

Python links

- “Extending and Embedding the Python Interpreter” (official Python 3 C API docs): https://docs.python.org/3/extending/
- Boost.Python: http://www.boost.org/doc/libs/1_63_0/libs/python/doc/html/index.html
- pybind11: https://github.com/pybind/pybind11/

Further References

- CppCon 2016: “Introduction to C++ python extensions and embedding Python in C++ Apps": https://www.youtube.com/watch?v=bJq1n4gQFfw
- This presentation: https://github.com/dermojo/presentations/
- ChaiScript (if you don’t like Python): http://chaiscript.com/