GROMACS - Task #2045

API design and language bindings

08/31/2016 05:47 PM - Peter Kasson

Status:	New
Priority:	Normal
Assignee:	Peter Kasson
Category:	
Target version:	
Difficulty:	hard

Description

We're gearing up for external API design. We ultimately want language support in Python and a C/C++ language. One question raised on the dev call is whether this is cleanest to do C-Python or C++-Python. Thoughts? Feel free to provide pointers to previous discussion as well.

Subtasks:

Task # 2698: gmxapi documentation integration	**New**
Feature # 2985: Python package documentation	**New**
Task # 3014: gmxapi example Python scripts	**New**
Task # 2877: use gmx::Options more	**New**
Task # 2893: Integrate gmxapi Python package	**Resolved**
Bug # 3144: gmxapi.mdrun does not clearly expose the output trajectory.	**New**
Task # 2894: Wrap importable Python code.	**Resolved**
Task # 2895: gmxapi Output proxy establishes execution dependency.	**Resolved**
Feature # 2896: Python packaging	**In Progress**
Feature # 2961: How should Python package find GROMACS resources under various circumst...	**New**
Task # 3027: Move sample_restraint development from GitHub to Gerrit	**In Progress**
Task # 3133: Cookiecutter for sample_restraint	**New**
Task # 2912: C++ extension module for Python bindings	**Resolved**
Feature # 2993: Scalar and structured type expression and definitions for API	**New**
Task # 3130: Interim handling of gmxapi data references.	**New**
Feature # 3140: Allow explicit input definition for gmxapi.operation function wrapper	**New**
Bug # 3141: gmxapi File placeholders missing from beta release	**New**
Feature # 2994: Data flow topology in gmxapi 2020	**New**
Bug # 3136: gmxapi.operation data flow topology unclear or incomplete	**New**
Feature # 3138: Improve ensemble support in Context specification.	**New**
Task # 3139: gmxapi Futures should be subscribable	**New**
Feature # 3147: gmxapi workflow checkpointing	**New**
Feature # 2996: gmxapi execution model	**New**
Feature # 3148: Roadmap for gmxapi filesystem interactions.	**New**
Feature # 3149: Python user interface for obtaining simulation artifacts as files.	**New**
Feature # 3152: Infrastructure and patterns for expressing public interfaces	**New**
Task # 3153: Let CMake process module directories earlier to support more modern CMake ...	**New**

Related issues:

Related to GROMACS - Feature #1625: Gromacs Python API	**New**
Related to GROMACS - Task #988: Definition of "public API"	**New**
Related to GROMACS - Feature #2585: Infrastructure supporting external API	**Resolved**
Related to GROMACS - Task #701: Add symbol visibility macros	**New**
Related to GROMACS - Feature #3034: Python gmxapi exception hierarchy	**Closed**

History

#1 - 09/02/2016 05:04 PM - Mark Abraham

For external APIs, historically the only robust thing to do has been to provide plain C bindings, because that's the only way to get a reliable **ABI**. This seems perfectly doable, we'd just need (a forest of) C functions that do

```
/* Some .h file */
extern "C" int gmx_getGlobalPositionCoordinates(gmx_State state, int size, gmx_CoordinateVector **x);

/* Some .cpp file */
extern "C" int gmx_getGlobalPositionCoordinates(gmx_State state, int *size, gmx_CoordinateVector **x)
{
  try
  {
    auto coordinatesHandle = static_cast<gmx::State>(state)->getGlobalPositionCoordinates();
    *size = coordinatesHandle->size();
    **x = coordinatesHandle->data();
  }
  catch (gmx::ApiException &e)
  {
    return convertToErrorCode(e);
  }
  return gmx_success;
}
```

which we'd likely have planned to have a layout useful so that a Python method can take the coordinate view that gmx_getGlobalPositionCoordinates() returns and e.g. copy it into a numpy array so someone can work happily. Or just use python array-lookup-like methods on the view if that's all they need.

A C++ API is technically feasible... but I understand (roughly) comes at the price of compiling both libgromacs and the calling code exactly the same way if something like a std::vector is in the function-call interface, because various compiler options do change the stdlib ABI, even if you use the same compiler on each side of the API. Same if such things are in the common header interface. (So, the API quickly starts looking pretty C-like.) Anyway, a C++-usable API probably requires also the work to establish what compiler things need to be the same on either side of the API (for each compiler we might support), and perhaps a template setup of a CMakeExternalProject build of libgromacs that allows such things to be coordinated without the prospective libgromacs coder needing to grok lots of such things. (We might still do such a thing even if the API formally has only C bindings.)

#2 - 09/04/2016 02:00 AM - Roland Schulz

- Related to Feature #1625: Gromacs Python API added

#3 - 09/04/2016 02:04 AM - Roland Schulz

Exactly the same is overstating it a bit. You want the same stl and you want the same c++ standard (c++11). Many other things don't have to be the same. Note that even with the old C api we had issues that we had a dependency on the config.h. So even with C one has to be careful. But I agree that C++ is more tricky but certainly doable. There are many C++ libraries out there.

Note that https://redmine.gromacs.org/issues/1625 has a lot of discussion of how to create a Python API. In particular about how to wrap C++ and whether a Python API should at all be a wrapper or a higher level abstraction.

#4 - 09/05/2016 04:58 PM - Mark Abraham

Roland Schulz wrote:

> Exactly the same is overstating it a bit. You want the same stl and you want the same c++ standard (c++11). Many other things don't have to be the same.

Indeed, but if std::vector is in the API, then one has to make sure all the defines are sufficiently similar. For example, you probably can't take translation units whose interfaces include std::vector and compile them with a mix of NDEBUG on/off and expect to link and run. I don't yet know how deep such rabbit holes are.

> Note that even with the old C api we had issues that we had a dependency on the config.h. So even with C one has to be careful. But I agree that C++ is more tricky but certainly doable. There are many C++ libraries out there.

Yes, there are many, and we should make sure we learn from suitable examples. In particular, there's probably not much we can learn from any header-only libraries.

#5 - 10/30/2016 10:42 PM - Eric Irrgang

I'm not sure I understand the ABI concern. If the suggested method for linking against libgromacs is with tools like find_package(GROMACS), then we already have a facility to convey compile-time details. Do we anticipate a user base of API developers that rely on binary distributions of libgromacs.so?

I anticipate that the development of C++ API elements can and will be a natural consequence of developing Python or other interfaces, which is one reason I am currently favoring pybind11.

#6 - 10/31/2016 05:36 AM - Teemu Murtola

Before jumping into technical implementation details, it would be good to have a clearly stated and commonly agreed goal for such external APIs. What should the user be able to accomplish through such APIs? One end of the spectrum is the expectation that the user can take essentially any part of Gromacs, reimplement that with modifications, and use that with the rest of the codebase. The other end is that the user is only able to call relatively high-level functionality (run this simulation, do this kind of analysis, ...). Somewhere in the middle is the ability to create plugins for some limited purposes (e.g., for custom analysis procedures or for applying extra forces during the simulation. If the user is expected to be able to replace substantial parts of mdrun through the API, what is the expected performance? On par with the native implementation, completely irrelevant, or something in between?

On the more technical side, a high-level decision would also be good on whether we want to have an idiomatic API for each language we support, or will we just blindly duplicate the APIs based on the smallest common denominator on what the different languages support?

#7 - 03/09/2017 07:28 PM - Peter Kasson

- Tracker changed from Bug to Task

- Difficulty hard added

- Difficulty deleted (uncategorized)

#8 - 03/09/2017 07:30 PM - Peter Kasson

Design work is ongoing. Eventually a draft API spec and before that a simple usage diagram will come here.
We're starting with some trajectory and TPR-modifying functionality (at the C++ level and a designed Python interface).

#9 - 03/13/2017 11:37 PM - Eric Irrgang

Addressing Teemu's comments, I think we can support several levels of API, but the highest levels will both take form first and require the most stability.

> Before jumping into technical implementation details, it would be good to have a clearly stated and commonly agreed goal for such external APIs. What should the user be able to accomplish through such APIs? One end of the spectrum is the expectation that the user can take essentially any part of Gromacs, reimplement that with modifications, and use that with the rest of the codebase. The other end is that the user is only able to call relatively high-level functionality (run this simulation, do this kind of analysis, ...). Somewhere in the middle is the ability to create plugins for some limited purposes (e.g., for custom analysis procedures or for applying extra forces during the simulation. If the user is expected to be able to replace substantial parts of mdrun through the API, what is the expected performance? On par with the native implementation, completely irrelevant, or something in between?

My near term goal for Python bindings would be to allow reimplementation of command-line tools to set up, run, and analyze simulations in terms of sensibly elemental tasks while providing higher-level convenience functions or "bottled" scriptlets for short series of tasks that may be currently handled by a single command-line tool. Important goals would be to replace filesystem and terminal I/O with API objects and to facilitate run->analyze->run workflows in which the input record, topology, and configuration are managed at the Python level.

I expect the more elemental tasks (reading a trajectory file, setting the box dimensions, etc) to be public C++ API functions translated to a low-level / core Python API with thin bindings sufficient to expose the functionality in a Pythonic way, while higher-level Python interface will likely be implemented in pure Python for transparency, convenience, idiomatic consistency, and stability. The appropriate C++ API functionality mostly either doesn't yet exist or requires updates/extensions, so I expect that while the high-level Python interface can probably be specified and stabilized first, the implementation of the Python bindings will coevolve with additions to the C++ public API and updates to the library API.

At the other end, the lowest levels of access I am thinking about right now are plugin interfaces for

- external potentials
- particle interactions
- inspection / analysis of running simulations
- custom biases, restraints, and free energy methods
- checkpointing and I/O managed by the Gromacs program context
- communication mediated by/with the Gromacs simulation loop

I think such code should be able to take advantage of Gromacs parallelism and optimizations.

> On the more technical side, a high-level decision would also be good on whether we want to have an idiomatic API for each language we support, or will we just blindly duplicate the APIs based on the smallest common denominator on what the different languages support?

Regarding whether to replicate API functionality across multiple supported languages, my vote would be a qualified "yes." I don't think development of public APIs for Python, C++, and even C or others should be necessarily encumbered by requiring progression in lock-step, though they should target the same functionality and be implemented in terms of the public C++ API where possible. Realistically, say, an aspect of the Python API could be implemented initially in terms of a lower-level interface when details for the C++ spec still need work. But to better maintain consistent semantics, maintainable (and maintained) code, and good testing coverage, implementing all user-facing interfaces in the highest possible API level seems like a good goal.

Furthermore, I think that we have the most opportunity to manage parallelism and exploit data locality if a distinction is made between Python API calls that operate on API objects (with data access and computation only happening at the C++ level) versus calls that explicitly bring data in or out of the Python interpreter for use with non-Gromacs tools. There is also a middle ground where the API could expose data across the bindings via the

Python buffer protocol, which gets a little bit trickier in terms of data ownership and the lifetimes of allocated memory, but which I think is an important mode of data access to design for from the beginning.

Regarding design documentation, it seems like the accepted process here is to submit proposed specifications as documentation updates in Gerrit. Would it be more appropriate to add to the Sphinx documentation for the dev-manual or to the doxygen pages for the API groups? The latter seems to be the current trend, but the former is more like the design documentation on the older website that hasn't been migrated.

Incidentally, regarding documents in the old webpage that haven't been reproduced in the Sphinx documentation or Doxygen manual, would it be helpful for me to incorporate them into new (proposed) pages or are they awaiting more careful curation?

On a technical note, and going back to the first comment, it is important to decide on goals / policies for ABI support as well as API support. Gromacs and many other scientific software packages are distributed as source rather than binaries, so the need to recompile plugins after building a new Gromacs doesn't seem that onerous, nor does requiring a plugin to be built with the same compiler and settings as extracted from FindGROMACS.cmake, but users who want to tie together tools from several projects into new code may find themselves with irreconcilable requirements. It may be acceptable to allow potential STL compatibilities, say, at the level of the shared objects and rely on Python (or a C API) for linkage, using raw pointers and/or copy conversion of incompatible data classes.

In other words, if there are compelling use cases for making it easy to link against a libgromacs.so whose build you don't control in order to interface it with someotherlibrary.so whose build you don't control, I am not (yet) aware of them, but it is worth establishing a clear policy in the API documentation.

#10 - 03/13/2017 11:43 PM - Eric Irrgang

- Related to Task #988: Definition of "public API" added

#11 - 07/24/2018 03:53 PM - Eric Irrgang

As an end goal, a GROMACS installation would have headers and a library sufficient to implement the modern and maintained tools accessible through the `gmx` binary. The C++ API would map to a Python API, which in turn would provide scriptability and management of more complex tasks. The API would provide sufficient abstractions to allow a sequence of API calls to be optimized for compute placement and data flow without unnecessary or unrequested I/O. A system installation of GROMACS should be sufficient to compile and link custom code that interacts efficiently with simulation and analysis code from the official release.

The scope of this and related issues can be refined into more discrete projects and milestones, which can be linked from this Issue.

- #2585 GROMACS repository and library infrastructure
- #988 definition of “public API”
- Data exchange: optimal and safe data movement or sharing across the API boundary
- Python API specification and reference implementation (prototype at https://github.com/kassonlab/gmxapi)

Some points for discussion in relation to the Python package:

- serves to prove libgmxapi functionality
- Currently tested with Travis-CI
- Ultimately, it should be available as part of a GROMACS installation
  - users may build and install package to a virtual environment of their choice from sources packaged in the GROMACS installation
  - should it be in the same repository and/or CMake build environment?
- Currently builds docs at readthedocs.org
- Debatable ongoing support for Python setuptools build and install
- Long term: should be able to build and install for a virtual environment from the GROMACS installation (or in reference to it)
- Not coupled to other GROMACS Python code. Will support Python 2.7 as long as feasible. (Arguably not feasible now...)
- pybind11 source bundled with repo
- Provides both an implementation and an API specification.
  - Researchers can be compatible with it without depending on it.
  - Writers of GROMACS extension code are free to use other Python bindings frameworks.
  - We provide tools and helpers, but C++ helpers use pybind11

#12 - 07/24/2018 03:57 PM - Eric Irrgang

- Related to Feature #2585: Infrastructure supporting external API added

#13 - 02/22/2019 11:30 AM - Gerrit Code Review Bot

Gerrit received a related DRAFT patchset '1' for Issue #2045.
Uploader: M. Eric Irrgang (ericirrgang@gmail.com)
Change-Id: gromacs~master~Idd72e9ede890f7fc97a680c5a5bffe97499eaaf5
Gerrit URL: https://gerrit.gromacs.org/9202

#14 - 02/27/2019 10:21 AM - Gerrit Code Review Bot

Gerrit received a related DRAFT patchset '1' for Issue #2045.
Uploader: M. Eric Irrgang (ericirrgang@gmail.com)
Change-Id: gromacs~master~Iae1518be21c37f0c6beec612da24c0adf93c9850
Gerrit URL: https://gerrit.gromacs.org/9238

#15 - 02/27/2019 12:13 PM - Gerrit Code Review Bot

Gerrit received a related DRAFT patchset '1' for Issue #2045.
Uploader: M. Eric Irrgang (ericirrgang@gmail.com)
Change-Id: gromacs~master~I71ec79eb5d8a6e4d61fb5d7d6e20176c9c07a6af
Gerrit URL: https://gerrit.gromacs.org/9241

#16 - 03/02/2019 01:38 AM - Eric Irrgang

- Related to Task #701: Add symbol visibility macros added

#17 - 03/02/2019 01:42 AM - Eric Irrgang

Q1/Q2 work under this issue supports the functionality described at https://github.com/kassonlab/gmxapi-scripts

Summary of recently posted Gerrit changes:

Work planning

- https://gerrit.gromacs.org/9202 Infrastructure for project design and development
- https://gerrit.gromacs.org/9240 Base change for acceptance criteria expressed with Jupyter notebook
- https://gerrit.gromacs.org/9241 Project goals / roadmap, and acceptance tests as Jupyter notebook cells.

Feature requests or functional requirements, depending on your perspective.

- FR1: https://gerrit.gromacs.org/9204
- FR2: https://gerrit.gromacs.org/9205

In particular, https://gerrit.gromacs.org/9241 is a first attempt towards establishing design documentation for the aspects of this issue to be addressed in the 2020-release development cycle.

#18 - 03/31/2019 04:53 PM - Eric Irrgang

Update to my comment from a month ago: this activity is now coordinated under issue https://redmine.gromacs.org/issues/2893 (and subtasks) and Gerrit changes tagged with topic "gmxapi".

#19 - 07/12/2019 11:22 AM - Eric Irrgang

- Related to Feature #3034: Python gmxapi exception hierarchy added