GROMACS - Feature #2896

Task # 2045 (New): API design and language bindings

Python packaging

03/18/2019 01:55 PM - Eric Irrgang

Status:	In Progress	
Priority:	Normal	
Assignee:		
Category:	build system	
Target version:	2020	
Difficulty:	uncategorized	

Description

Sources for GROMACS Python interfaces and extension development tools are targeted for inclusion in the GROMACS 2020 release. This issue serves to track details about source organization, installation scenarios, and Python packaging.

Criteria for completion:

- Enumeration of (intended) supported (and unsupported) installation scenarios for GROMACS Python package(s).
- A roadmap of support for installation scenarios and packaging schemes.
- Policies or guidelines for repository layout, integration testing, documentation schemes, and build system schemes for centralized gmxapi sources.

It seems like this should be in the form of a versioned document and/or cross-linked to iterations on installation documentation. Thoughts?

Initial discussion:

Not all of the following goals can be met for the 2020 release, so they should be discussed and prioritized in the resolution of this issue.

- *As a researcher, I want to access GROMACS by importing a Python module so that I can integrate my Python-driven tool set.*
- *As an HPC user, I want to access the GROMACS installation on my cluster by importing a Python module so that I can use the exact same GROMACS build from Python or the CLI.*
- *As a researcher, I want to easily rebuild the GROMACS Python extension so that I can have native performance from different Python installations or "virtual environments".*
- *As a Python user, I want to easily find/build/install the package into my Python environment so that I can get up and running quickly.*
- *As a core developer, I want (continued) CMake-driven building and testing so that I don't need to worry about Python or its packaging details.*

  Current thoughts re: the above are that the C++ extension in the Python package should be built against an existing GROMACS installation at the time the package is installed into a user's Python environment. The easiest way to allow this would be with a source distribution in the GROMACS installation path, but a Python package hosting service could also provide a source distribution that would build locally (with the user sourcing GMXRC first or something). To the extent that GROMACS supports or encourages binary distributions, binary Python distribution packages can also be made available, but each packaging system and installation case should be considered in the context of MPI and CUDA support and with a balance of ease-of-installation versus accessibility of full native performance.

- *As a core developer, I want robust automated CI testing so that C++ code changes are verified to function correctly through both CLI and Python interfaces.*
- *As an external developer, I want to write extensions that can be built against GROMACS installations and used in gmxapi scripts so that my efforts serve a wider audience sustainably.*
- *As a system administrator, I want to be able to build, test and install all GROMACS components at once, with the same minimal dependencies, so that I can manage an HPC software module system.*
- *As a system administrator, I want the option of building and installing the GROMACS Python package more than once so that I can manage my support of loadable Python environments with minimal coupling to my GROMACS installations.*

- *As an HPC user or administrator, I want the GROMACS Python package to be cross-compilable with a CMakeToolchain that is created or informed by the GROMACS installation so that I can use the package in environments with disparate login and compute node architectures.*

- *As an HPC administrator, I want the GROMACS Python package to avoid run-time dependencies so that my filesystem server isn't swamped by dynamic library loads when large jobs are launched.*

  We can build a shared object library for an importable Python module that is statically linked against a GROMACS library archive compiled with position-independent code at an earlier time so that the Python executable only dlopen a single .so. With that, it is only a few extra CMake lines to get help from scikit-build to allow admins to compile the package into a Python interpreter, but then we impose (and require) the full GROMACS execution environment on a system-wide Python interpreter, which will not be desirably in more heterogeneous computing environments.

Subtasks:

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**

Related issues:

Related to GROMACS - Task #701: Add symbol visibility macros	**New**
Related to GROMACS - Task #2912: C++ extension module for Python bindings	**Resolved**
Related to GROMACS - Feature #3038: Improvements to MD plugin development env...	**New**

Associated revisions

Revision ad2ac649 - 04/24/2019 03:02 PM - Eric Irrgang

Add Flake8 linting to gmxapi tests.

Add the flake8 package to requirements-test.txt and a run_flake8
docker entry point. This change does not require Python source
checking, but facilitates it.

Refs: #2896
Refs: #2615

Gerrit patch set 10419/1

Revision 354da806 - 04/24/2019 03:31 PM - Eric Irrgang

More Python testing infrastructure for gmxapi.

- Add and improve test fixtures.
- Generate MD input files for tests.

Refs: #2756
Refs: #2896

Gerrit patch set 9392/2

Revision 88db82f5 - 04/24/2019 03:51 PM - Eric Irrgang

[RFC] CMake front-end for Python packaging machinery.

Refs: #2896

Gerrit patch set 9378/2

Revision 929343d1 - 05/09/2019 01:12 PM - Eric Irrgang

More Python testing infrastructure for gmxapi.

- Add and improve test fixtures.
- Generate MD input files for tests.

Refs: #2756
Refs: #2896

Change-Id: I15ac8e44844cbf699cfe362e2fa443d07a355b3d

Revision 17454646 - 05/17/2019 05:42 PM - Eric Irrgang

CMake front-end for Python packaging machinery.

Refs: #2896

Gerrit change 9378/3

Revision a29abd63 - 07/09/2019 11:30 AM - Eric Irrgang

Begin CMake front-end for Python packaging machinery.

Allow gmxapi Python package sources to be discovered in the main GROMACS
CMake configuration step. During build, a functioning package will
appear in the build tree in
${CMAKE_BINARY_DIR}/python_packaging/src/gmxapi_staging that can be used
for tests and auto-generated documentation builds.

A Python 'sdist' can be built from this location in the build tree, or
the package can be built and installed directly from the source tree
with setup.py in python_packaging/src/

Refs: #2896

Change-Id: I225dd890c93405880ef057465eb7c159cd097665

History

#1 - 03/19/2019 10:56 AM - Mark Abraham

Eric Irrgang wrote:

> Sources for GROMACS Python interfaces and extension development tools are targeted for inclusion in the GROMACS 2020 release. This issue serves to track details about source organization, installation scenarios, and Python packaging.
>
> Criteria for completion:
>
> - Enumeration of (intended) supported (and unsupported) installation scenarios for GROMACS Python package(s).
> - A roadmap of support for installation scenarios and packaging schemes.
> - Policies or guidelines for repository layout, integration testing, documentation schemes, and build system schemes for centralized gmxapi sources.
>
> It seems like this should be in the form of a versioned document and/or cross-linked to iterations on installation documentation. Thoughts?

Seems fine. We should start here, and end up with a checklist somewhere that individual commits can resolve.

I suggest as priorities

- *As a researcher, I want to access GROMACS by importing a Python module so that I can integrate my Python-driven tool set.*
- *As a Python user, I want to easily find/build/install the package into my Python environment so that I can get up and running quickly.*
- *As a core developer, I want (continued) CMake-driven building and testing so that I don't need to worry about Python or its packaging details.*
- *As an external developer, I want to write extensions that can be built against GROMACS installations and used in gmxapi scripts so that my efforts serve a wider audience sustainably.*

But I would note that we should want to converge our end-to-end testing on the Python API sooner rather than later, to help dogfood the API and reduce duplication of testing code.

#2 - 03/19/2019 01:39 PM - Eric Irrgang

An additional consideration: to minimize installed sources, we might want to rely on native packaging systems even for the pybind11 header library. For each packaging system we enable, we should consider the best way to satisfy the pybind11 header library dependency and which way(s) to support.

However, since the pybind11 header library provides implementation code rather than public interfaces, I believe it is conventional and recommended for projects to bundle pybind11.

I think this question is only relevant in the case where a binary distribution of GROMACS includes the source distribution of the gmxapi Python package, which we might just treat as a special case when we enable those packaging targets, rather than try to solve generally.

#3 - 03/22/2019 11:45 AM - Eric Irrgang

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

#4 - 03/22/2019 11:48 AM - Eric Irrgang

Note: If the Python module will statically link the GROMACS library, there should be a GROMACS library static linking target with position-independent code and default hidden symbol visibility.

#5 - 03/31/2019 05:25 PM - Eric Irrgang

- Related to Task #2912: C++ extension module for Python bindings added

#6 - 04/01/2019 01:54 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #2896.
Uploader: M. Eric Irrgang (ericirrgang@gmail.com)
Change-Id: gromacs~master~I225dd890c93405880ef057465eb7c159cd097665
Gerrit URL: https://gerrit.gromacs.org/9378

#7 - 04/03/2019 12:57 PM - Gerrit Code Review Bot

Gerrit received a related DRAFT patchset '1' for Issue #2896.
Uploader: M. Eric Irrgang (ericirrgang@gmail.com)
Change-Id: gromacs~master~I15ac8e44844cbf699cfe362e2fa443d07a355b3d
Gerrit URL: https://gerrit.gromacs.org/9392

#8 - 07/17/2019 02:15 PM - Eric Irrgang

- Related to Feature #3038: Improvements to MD plugin development environment added

#9 - 09/25/2019 01:41 PM - Eric Irrgang

- Status changed from New to In Progress

#10 - 10/09/2019 04:08 PM - Eric Irrgang

Progress update

Organized by the scenarios highlighted by Mark, but also addressing the others from the issue description.

As a researcher, I want to access GROMACS by importing a Python module so that I can integrate my Python-driven tool set.

The primary install cases are now documented as installation with pip, either from the GROMACS source repository or from the default pip behavior of retrieving a package from PyPI.org. This results in the package being available for import by the same Python interpreter that ran the pip command. For user-space installs, a Python venv is recommended so that the default installation path is to a directory writable by the user.

A use case that has not been pursued is a unified build/install of a Python package that bundles the GROMACS library internally and is not tied to a traditional GROMACS installation. If this is an interesting use case to you, please consider some additional questions: Should the GROMACS installation be accessible through the Python package installation? Should we move the setup.py to the top level of the repository and make it an alternative entry point for GROMACS installations?

Binary packaging has not had much consideration because of the perceived trade-offs of difficulty (combinatoric complexity for performance and compatibility) versus utility (are there substantial binary package use cases?). (Feedback welcome.)

As a Python user, I want to easily find/build/install the package into my Python environment so that I can get up and running quickly.

The initial proposal was to install a Python source distribution with GROMACS so that a user of a GROMACS installation would have a clear path to installing the package in their preferred Python environment (sort of an alternative or additional step to doing source GMXRC or module load gromacs), but this was rejected as being messy and/or not particularly helpful.

The gmxapi package is searchable online or via pip, but is not particularly obvious in the GROMACS source or web site. However, if a user browses to the "gmxapi" section of the manual, they should quickly find the installation cases and requirements.

Solutions pursued so far have assumed that the primary target user is an HPC user with a system-provided GROMACS installation. In this case, it is unlikely (or a lot of work to ensure) that a GROMACS installation is compatible with the user's preferred Python installation, so we assumed that we would need to focus on helping users to build the Python package themselves from a source distribution against the chosen GROMACS and Python installations.

As a core developer, I want (continued) CMake-driven building and testing so that I don't need to worry about Python or its packaging details.

We are using scikit-build to move as much of the package build as possible into the CMake regime and to stage the built extension module with a copy of the pure Python components so that an importable package is produced in the build tree.

Setting GMX_PYTHON_PACKAGE=ON tells CMake to recurse into the python_packaging directory, building and staging the Python package for testing and automated docstring extraction.

As an external developer, I want to write extensions that can be built against GROMACS installations and used in gmxapi scripts so that my efforts serve a wider audience sustainably.

The sample_restraint package is built and tested when GMX_PYTHON_PACKAGE=ON, but needs some updates to make it as easy to use as a template for new extensions as when it was a standalone repository. I think a cookie cutter or other skeleton repository referencing the python_packaging/sample_restraint subtree will solve this in the near future.

It can be challenging to configure the build system for an external project to be robustly compatible with a GROMACS installation, especially on systems with multiple MPIs, C++ compilers, stdlib implementations, Pythons, etc. One possible remediation is to provide a CMakeToolchains file with the GROMACS installation, but there may be other/better solutions. As of this comment, I am not aware of any GROMACS-driven CI tests that trigger GROMACS client software tests, but the gmxapi and sample_restraint package tests will provide some coverage in the near future. We may also hope for feedback from nightly CI tests from external projects with GROMACS dependencies.

beta status

Most of the gmxapi documentation is not built with the 2020 beta 1. As of this writing, skeletal documentation and installation instructions are included in the manual for the cases described above.

I think we can improve the documentation entry points. I will be submitting some documentation bug fixes. There has not been much feedback on the layout of current documentation stubs, so I will be fleshing out these as well.

I welcome feedback on time line and release targets for additional / altered installation cases.

Additional Ideas from Python infrastructure

I've noticed a few gizmos in various infrastructure from Python itself that might inspire helpful features in the GROMACS build system. Thoughts on the following?

- A make venv target to provision a Python virtual environment, such as for documentation builds or testing.
- A bootstrapping module, like the ensurepip package in Python >=3.4, written in pure Python, that can be installed with GROMACS to help users configure gmxapi (and a Python venv) for their Python interpreter.