GROMACS - Task #653

Task # 652 (Blocked, need info): Change selection method implementation to use C++

Extend options module for use in the selection engine

01/09/2011 04:56 PM - Teemu Murtola

Status:	Closed
Priority:	Low
Assignee:	Teemu Murtola
Category:	core library
Target version:	2018
Difficulty:	

Description

The user-visible interface of the options engine should already support most of the features required by the selection engine; the main thing on this side is to implement options that return index groups and positions in addition to the currently supported basic data types. This should be straightforward once the issue below is implemented.

The difficult part is to implement a mechanism that allows assignment of option values from other types than strings. It should support:

- Assignment from selection evaluation tree elements (currently t_selelem objects), which should initialize the evaluation tree element such that it will write its evaluated value to the memory location set for the option value.
- Assignment from basic data types (at least numeric types). Can be implemented as a special case of the above (if the value of the tree element is constant and known, it can be directly written to the option value).
  Both issues take some careful thinking on how to make everything type-safe (some run-time type checking will be necessary).

The division of responsibilities between the options and the selection modules should be carefully though about to not introduce a circular dependency between the two modules.

Associated revisions

Revision fd73b6f7 - 08/18/2016 02:35 PM - Teemu Murtola

Support for non-string option value assignment

Make it possible to assign non-string values to options through
OptionsAssigner. For now, the only thing that actually works is
assigning std::string values, and assigning the exact type stored for
the option. Also, the list of supported conversions is initialized
separately for every instance of each option, but in practice that might
not matter that much.

This adds the infrastructure that allows assigning option values from
something like a JSON structure that already does data type conversions
during parsing. The same infrastructure can also support #653.

Change-Id: I8f81d57da1b6eeac06a6ac9a8ab68bfa8ed0c148

History

#1 - 02/18/2011 10:19 AM - Teemu Murtola

I think I now have a design sketch that allows all these issues to be implemented as extensions on top of a basic selection engine (can explain more details if someone is interested in implementing it or otherwise). This will require more or less full-blown RTTI, and would greatly benefit from boost::any. So there are some policy decisions to be made before it can be implemented.

For RTTI, even the existing C implementation for selections uses a crude hand-coded RTTI in form of enums and unions, so using the facilities provided by the compiler for doing all that would be quite attractive.

For a truly extensible implementation (such that the options module does not need changes when adding new supported data types), something like boost::any will be essential. I would feel that it's better to use the solution from boost than to write a more or less identical implementation... In particular, because boost::any is quite self-contained, and only pulls in a few utility headers from the rest of boost.

#2 - 12/29/2012 01:57 PM - Teemu Murtola

- Target version set to future

#3 - 04/30/2013 05:42 AM - Teemu Murtola

- Status changed from New to Accepted

#4 - 04/30/2013 05:42 AM - Teemu Murtola

- Status changed from Accepted to In Progress

#5 - 04/30/2013 05:44 AM - Teemu Murtola

- Status changed from In Progress to Blocked, need info

Needs a decision on the use of boost::any. It will be hard to implement this without something like it. There is no policy on how to propose additions to the allowed set of boost headers, so waiting for that.

#6 - 04/30/2013 01:05 PM - Mark Abraham

http://www.boost.org/doc/libs/1_53_0/doc/html/any.html does look like it would be useful enough for selection and/or option machinery to be worth allowing.

Is there a GROMACS user-space example you can give for something useful that is much easier to implement with boost::any than a home-grown work-alike? That will make it much easier for those of us less expert in the existing selection machinery to understand the problem.

How can we see the dependency tree of a Boost module?

Is "it works on the older compilers used by Jenkins" an acceptable proxy for "it probably won't break under vendor X's antiquated C++ compiler fork?"

#7 - 04/30/2013 08:35 PM - Teemu Murtola

Mark Abraham wrote:

> Is there a GROMACS user-space example you can give for something useful that is much easier to implement with boost::any than a home-grown work-alike? That will make it much easier for those of us less expert in the existing selection machinery to understand the problem.

It will be relatively easy to just copy the implementation for boost::any and use it; the essential parts are only a few tens of lines, so using a home-grown implementation is never going to be hard. But I don't think that it gains much, either, and a boost alternative is probably more maintainable in the sense that it is more well-known.

Where I would like to use it is to replace most of SelectionValue from source:src/gromacs/selection/parsetree.h, and to have a gmx::OptionsAssigner::appendValue(const boost::any &value) (from there, it propagates into the options implementation internals, but this should give you the general idea). This would replace code from source:src/gromacs/selection/params.cpp, which is now full of switch statements based on the type of the input variables. It may result in somewhat more code, but the end result should be much more structured.

> How can we see the dependency tree of a Boost module?

For this case, it is very easy: just look at the boost/any.hpp header to see that it has no dependencies (except the boost configuration headers and some dependencies on type_traits that we can avoid if we remove some of the any_cast implementations).

> Is "it works on the older compilers used by Jenkins" an acceptable proxy for "it probably won't break under vendor X's antiquated C++ compiler fork?"

Probably, in particular if we avoid those any_cast implementations that use some fancy stuff.

#8 - 05/22/2013 06:02 AM - Mark Abraham

- Target version changed from future to 5.0

#9 - 09/15/2013 07:09 AM - Teemu Murtola

- Target version changed from 5.0 to future

Not likely going to happen for 5.0, for the same reason as the parent task.

#10 - 06/10/2014 03:01 PM - Teemu Murtola

- Category set to core library

#11 - 07/12/2016 06:43 AM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #653.
Uploader: Teemu Murtola (teemu.murtola@gmail.com)

Change-Id: I8f81d57da1b6eeac06a6ac9a8ab68bfa8ed0c148
Gerrit URL: https://gerrit.gromacs.org/6036

#12 - 08/03/2016 04:51 PM - Teemu Murtola

- Status changed from Blocked, need info to Fix uploaded

- Assignee set to Teemu Murtola

- Target version changed from future to 2018

#13 - 04/23/2017 12:09 PM - Mark Abraham

Is more work needed here? Bump to 2018 version?

#14 - 04/23/2017 08:05 PM - Teemu Murtola

- Status changed from Fix uploaded to Closed