GROMACS - Task #856

Task # 827 (Closed): Improve interfaces in analysisdata, selection and trajectoryanalysis modules

Improve public interface of gmx::Selection

12/22/2011 10:03 PM - Teemu Murtola

Status:	Closed
Priority:	Normal
Assignee:	Teemu Murtola
Category:	selections
Target version:	5.0
Difficulty:	uncategorized

Description

Currently, the Selection class contains quite a few methods like

```
  rvec x(int i) const;
  real mass(int i) const;
  ...
```

I think it would be better to make the interface less flat to better describe the structure of the data, i.e., that a selection is made of a set of positions, which then have some properties. There are also some selection-level properties. This division would make the class more intuitive to use.   There are several possible solutions that I can imagine. All have a common feature: a lightweight wrapper class that looks conceptually like this:

```
class Selection::Position
{
    public:
        rvec x() const;
        real mass() const;
        ...

    private:
        Selection *sel_;
        int  index;
};
```

This wrapper class provides methods to access all properties of one single position, and just reads them from the underlying data structures in Selection. I can see two basic alternatives on how this can be used:

1. Provide methods

   ```
     int positionCount() const;
     Position position(int i) const;
   ```

   in Selection.
2. Provide a container or a look-a-like container:

   ```
     const PositionList &positions() const;
   ```

   This approach further can be divided based on what PositionList is:
   1. PositionList is std::vector<Position>. A similar alternative is to redefined Position such that it contains all the necessary information, but then it is necessary to duplicate some information from the internal gmx_ana_pos_t structure.
   2. PositionList is a custom non-STL container.
   3. PositionList is a custom container that tries to behave as std::vector<Position> where possible, but only stores the Selection pointer and the index in iterators, not with every element.

I'd like some comments on which alternative would be the best before I start implementing one of them. I personally prefer either 1 for its simplicity or then 2.3, but I wouldn't want to start implementing 2.3 only to find it rejected in review due to its complexity. On the other hand, if 2.3 is the way to go, I wouldn't want to first implement 1 and convert call sites more than once. I'll try provide more thoughts on the advantages and disadvantages of these options later, but let this serve as a conversation starter.

Associated revisions

Revision d8784e9e - 02/22/2012 06:38 AM - Teemu Murtola

More structured Selection interface.

Improved Selection interface as described in issue #856.
Also changed tests to use the new interface more naturally, which
required regenerating the XML files.

Change-Id: I999fa2f944a4a10ffaefc570a1525a88d3fd4824

History

#1 - 12/27/2011 12:20 PM - Roland Schulz

I agree that of the different 2 options 2.3 is best. Whether 2.3 or 1 is better depends on how much more effort 2.3 is. I think it has only slight advantages (e.g. STL algorithms support) so I would think it is only worth the effort if it is not much more work.

#2 - 12/27/2011 01:43 PM - Teemu Murtola

Option 2.3 requires quite a lot of code (more than any of the other options), but most of that should be straightforward to write. And yes, the advantages wouldn't be that big, mostly enabling STL algorithms and ability to write loops using iterators.

The main issue why I didn't directly go with 2.3 is that it's not possible to implement such a container to fulfill all the requirements of STL sequence containers, although in practice it would probably work with all non-modifying STL algorithms (which are the only relevant ones, since the Selection class is not intended to modify the selections). The issue is that the standard requires that operator*() returns a true reference (Position & in this case) for all but input/output iterators, which is not possible to implement except like in 2.1, which defeats the purpose. So the iterators would strictly be only input iterators, although to the user they would appear to work like random-access iterators.

#3 - 01/25/2012 05:02 PM - Teemu Murtola

- Status changed from New to Feedback wanted

Change according to option 1 is now pending review in gerrit (https://gerrit.gromacs.org/431).

#4 - 03/16/2012 09:13 PM - Teemu Murtola

- Status changed from Feedback wanted to Closed

Has been merged.

#5 - 02/14/2014 08:08 PM - Teemu Murtola

- Project changed from Next-generation analysis tools to GROMACS

- Category set to selections