GROMACS - Feature #1842

Replace XML with JSON

10/12/2015 12:45 PM - Erik Lindahl

Status:	New	
Priority:	Normal	
Assignee:		
Category:		
Target version:		
Difficulty:	uncategorized	

Description

Struggling with libxml2 on windows during the weekend made me think deeper about our file formats.

I guess the good news with dragging our feet and not implementing a new topology format is that we haven't done a lot of work in vain :-)

We know the obvious arguments for XML, but lately I've increasingly come to agree with the ones against it, in particular that it is a royal pain to edit manually and that libxml2 is not as portable and easy to use as one could hope.

So, without further ado: Should we move to JSON instead? It's much easier to work with, and there are plenty of free (BSD), fast and portable implementations that are relatively small headers, so we could ship it as part of Gromacs.

David has added a small amount of data files in XML, but those should be trivial to move. The question is how difficult it would be to alter the goole test reference data?

Related issues:

Related to GROMACS - Task #2017: Modularize simulation option storage and rea...	**In Progress**
Related to GROMACS - Task #2383: Add JSON interface to write and read files	**New**

History

#1 - 10/12/2015 01:37 PM - David van der Spoel

www.w3schools.com/json/
The example is not compelling.

I think the "manual editing" argument is not valid. These files are for computer reading and writing only. All we ever discussed is replacing tpr files by xml, not top files even though they are not perfect either.

On a side not I have added lots of XML in other branches, but if we adopt the idea of having a gromacs layer between application code and storage then a change of library from xml to json would be relatively easy. I initiated a discussion on gmx-developers few weeks ago after Teemu suggested this layer, but not many people responded.

#2 - 10/12/2015 02:14 PM - Erik Lindahl

HI,

We've certainly discussed new formats both for mdp and top on several occasions. We do need something that is more validated and strict, and for both of these is would be a major PITA to use XML.

One huge advantage of JSON appears to be the implementation size. For example, the RAPIDJSON implementation (https://github.com/miloyip/rapidjson) is less than 10,000 lines of header files in total, and could easily be shipped with GROMACS.

#3 - 10/12/2015 02:20 PM - Erik Lindahl

PS:

As you say, we should ideally have an interface layer that is agnostic to the actual format. I think that would be relatively straightforward if we stick to a simple hierarchical structure of nodes/contents, and avoid using e.g. attributes in XML.

#4 - 10/12/2015 06:23 PM - Roland Schulz

Of course we have one less dependency if we use the same format for data-exchange (only computer processed) and input/config files. But it wouldn't give us the best format for each because the requirements are quite different:

data-exchange:

- highly portable (tpr needs to be read everywhere)
- efficient and scalable for large files (we should be able to read any subset of a tpr without having all in memory)
- easy to post process (xslt is used for refdata)
- easy to process by other programs

config-format:

- portable less important (mdp, top, ..) are only read by grompp
- easy to read and write by humans important
- some form of embedded logic would be nice (#1744)

So it might be best to stick with xml for data-exchange and use e.g. Python for the config format. But even with XML (without extensions) it isn't possible to fast seek. So for tpr it might be better to keep using a binary format (maybe TNG based). Without tpr the requirements on the data-exchange format are less and possible easier to combine with the config-format. But xslt isn't supported by any of the reasonable alternatives.

#5 - 10/12/2015 06:34 PM - Erik Lindahl

My increasing concern is that XML (at least libxml2) fails several of the requirements for the data exchange too. It does not compile out of the box on most "special" computers. Even though I eventually got it working on K and Power8, I suspect a novice user won't. On windows all the downloadable binary versions I tried failed until I went into the code and started altering defines in the headers. DOM isn't scalable for large files, and with SAX there is a lot of manual work to assemble everything.

So, in particular as we will likely use data exchange more, I think there are only two options for this part:

1) Find a small stand-alone portable version of XML we can include in our source.
2) Ditch XML.

How difficult would it be to replace the XSLT parts we use in the refdata right now?

Or, alternatively, does anybody have experience with a small portable free XML library?

Cheers,

Erik

#6 - 10/12/2015 06:58 PM - Roland Schulz

I can't remember having any problems with libxml2 on Windows.

If we want both DOM and SAX we probably won't find a small library.

There is no obvious way for me to reimplement the refdata browser representation without xslt. While it might not be the most important feature, it is very discouraging to invest time on it and then having it removed because a decision is reversed (7 years after it is first discussed).

Currently XML isn't a hard requirement and if we don't use it for tpr it wouldn't become one for mdrun.

#7 - 10/12/2015 07:18 PM - Erik Lindahl

The binaries linked from xmlsoft.org do not work on a vanilla Windows10 + MSVC2013 or MSVC2015 installation; it took me a while to find out that you need to alter the defines for ICONV support in their headers.

I too am not thrilled with changing, but originally our hope/expectation was that XML would gradually become a core standardized component that is always available out-of-the box both on all platforms. Unfortunately this has not happened - there are probably many reasons why.

There's no question it's a royal pain (not to mention piss-off) to have to reimplement stuff (ask me, I'm getting seriously tired of SIMD implementations :-), but it's also obvious that if we're not really happy with a solution we will eventually replace it, and if that is the case I would rather do it sooner so we don't sink even more time & effort into something that will eventually have to go.

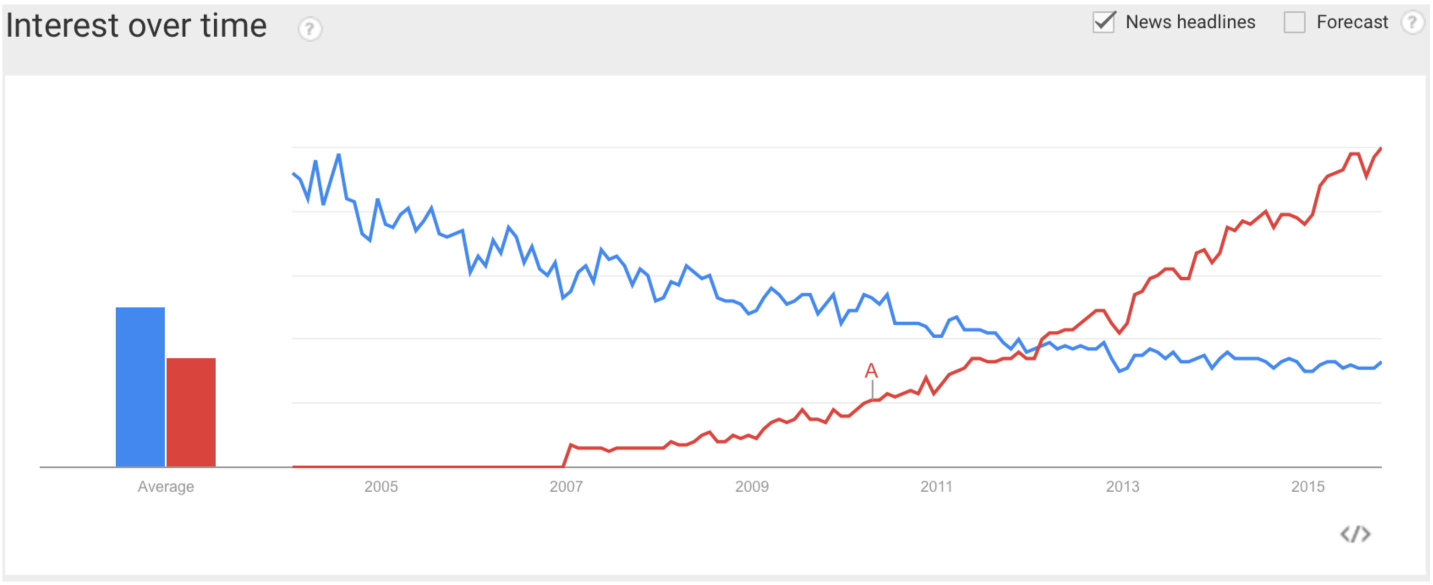
Limiting XML to only the refdata seems a bad solution (since we already need other formats), and at some point we will need something to describe e.g. tpr files, not to mention that we are increasingly adding more and more data formats, so before we know it whatever format we use will be quite a firm requirement - hoping that it won't be a problem seems futile.

Again, I don't have a patent solution, but I think the realistic long-term alternatives we have are to either make sure XML always works (by having a small version included), or going with something else?

#8 - 10/12/2015 07:27 PM - Erik Lindahl

- File xml_json.png added

The attached image might say more than thousand words:



Right now we're betting on blue.

#9 - 10/12/2015 07:57 PM - Teemu Murtola

> There is no obvious way for me to reimplement the refdata browser representation without xslt. While it might not be the most important feature, it is very discouraging to invest time on it and then having it removed because a decision is reversed (7 years after it is first discussed).

There is currently about 500 lines of XSLT for inspecting the data. This could probably be replaced with some Python script that can convert JSON (or whatever the format is) into HTML and open the result in a browser, but that takes some effort (and likely, a lot more code) to get to the same level of usability as the current XSLT-based approach.

Other than the XSLT, changing the storage format for reference data would probably be relatively easy; the only thing fancy thing that the XML implementation does is that it puts a bit of effort into formatting the XML files such that there is not too much unnecessary stuff there, that text-format diffs make sense, and that large blocks of multi-line text are easy to check directly from the XML file.

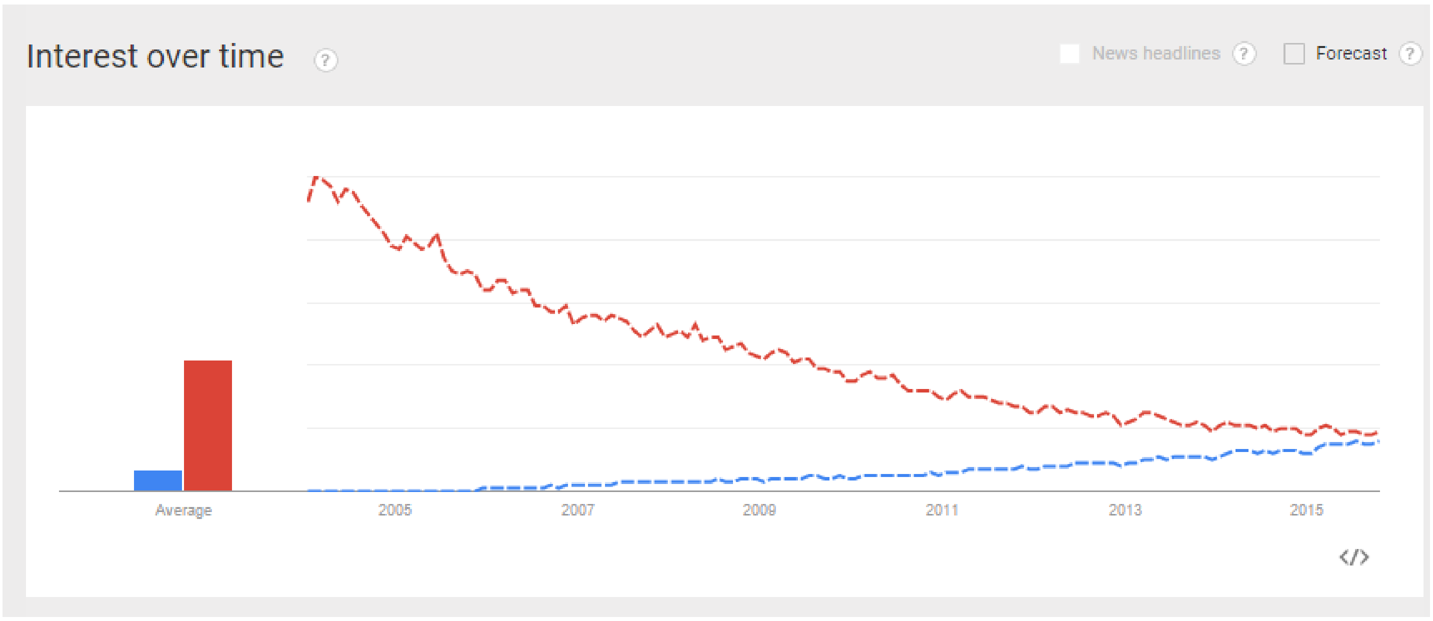
#10 - 10/12/2015 08:33 PM - Roland Schulz

- File json_xml.png added

I think before we can decide on what format is best we should list the requirements.
Questions I have:

- What requirements did I forget in my list?
- Do we want to have 1 format, 2 formats (human input and data exchange or text and binary) or 3 formats (human input, data exchange, and binary)?
- What's the plan for tpr? Text based would be a problem if we want to do 1 billion particles (e.g. for material science).
- What's the plan for input formats? What's the plan for #1744?

How do you get that data? I get



which both seems level over the last year with json being lower.

#11 - 10/12/2015 08:48 PM - Roland Schulz

An xslt replacement might be something like http://twigkit.github.io/tempo/. Or one could just convert to XML and then with the existing XSLT to HTML. This could be done with either Python or JavaScript (in browser). json to xml exists for Python and JavaScript.

#12 - 10/12/2015 09:06 PM - Erik Lindahl

Oh, I just followed a link from stackexchange - I think that trend was for XML API vs JSON API which should be more focused on the programmers compared to XML vs JSON which I guess is more focused on users. However, even the latter has XML decreasing exponentially and JSON predicted to overtake it early 2016. I actually care less about the JSON curve, but increasing our investments in a standard with decaying popularity seems like a battle we'll eventually lose.

As for requirements I can think of:

- It would be good to have a higher-level abstract interface layer in GROMACS, under which we can implement whatever data format will be popular in 5-10 years. By necessity, this might require us to avoid using all the bells and whistles of any format.
- For the foreseeable future I think we need both a human-readable/editable text format and a binary format, so it would be good if we can avoid splitting the first one further. Ideally we could have a binary format that corresponds directly to the text-based format, but that might be difficult: A tpr file is not just a binary version of the top file.
- It is good if the text-based format can be written reasonably efficiently in a text editor (not a deal-breaker, but a drawback for XML - editing is one thing, but writing XML from scratch is a pain)

I don't have any particular experience of flow control in these languages, but I guess we could always introduce such concepts directly in our data structures if necessary - this would also be a way to avoid being entirely dependent on a specific format.

#13 - 10/12/2015 09:38 PM - David van der Spoel

We should be careful not to mix things here. Talk about 1 or 2 formats is not very realistic looking at share/gromacs/top. We have tons of different file formats and will need to support those for a long time. Dumping everything (e.g. a whole force field directory) into one file will not likely make our life easier and introduce a ton of bug.

What we should discuss in this redmine IMHO is only what kind of dependency we can add to mdrun.

#14 - 10/12/2015 09:57 PM - Roland Schulz

Yes, we will have to support the legacy formats for user input files (could be done by a converter). But it does make sense to agree on what we want as format for all new/updated formats. We could convert all files in share/gromacs/top to XML/JSON/YAML/.... Of course each file would have a different type/schema. No one has suggested to put them all in one large files. This is just about the format (not even schema/type).

For tpr, instead of making it directly text/human-readable, I suggest we make it possible to convert tpr back and forth to a text representation. The text files should overlap as much as possible with the input files to grompp. Most files could be the same we would only need an extra text file for the grompp prepocessing data (e.g. constraints).

#15 - 10/13/2015 06:26 PM - Teemu Murtola

Roland Schulz wrote:

> An xslt replacement might be something like http://twigkit.github.io/tempo/.

If we give up the principle that all non-trivial scripting should use a single language that most people understand, we can probably do a lot of stuff with JavaScript on this front. But without a significant amount of effort, it won't be as usable as the old XML approach, where you could point your browser to the directory containing the XML files, and click on the XML file and get it automatically formatted. And an extra complication is that we want to run all the stuff from the file system, while a lot of more complicated stuff requires a backing HTTP server by default...

> Or one could just convert to XML and then with the existing XSLT to HTML. This could be done with either Python or JavaScript (in browser). json to xml exists for Python and JavaScript.

These all require converting the files to temporary files, before opening them in the browser, which in turn requires extra supporting infrastructure to make it easy to manage this extra set of files. Also Tempo linked above probably requires separate HTML pages, or quite a bit of extra JavaScript code that dynamically allows one to select the JSON source file from the browser...

#16 - 10/13/2015 07:12 PM - Roland Schulz

I think in Javascript we wouldn't need any temporary files. Both x2js and xslt can be done on strings. But of course we would need a way to pick the json file because we couldn't embed the javascript the same way as xslt. So I agree we would either need to generate an html for each json (could be very simple - but still complicates cmake), have a file picker in javascript (I agree we don't want much non-Python scripting), or have a Python script which generates a temporary html and opens it in the browser. I think a temporary html generated by a Python script (e.g. called view_json) would be best. It could either do the json->html (direct or indirect through xml) or just create a html with the javascript to do it in the browser. One couldn't just click on the file but would need to call the script. But given that all other GROMACS tools are command line based that's probably fine.

#17 - 10/13/2015 07:20 PM - Erik Lindahl

Roland: I haven't had a chance to look at your XSLT yet; what is the biggest advantage of being able to view the files in a browser?

#18 - 10/13/2015 07:34 PM - Roland Schulz

This is all Teemu's work. It is there to visual reference data if it isn't easy to compare as XML text. Just open any xml in Firefox (doesn't work with Chrome or IE) for which the folder contains an xsl file (e.g. selection).

#19 - 10/13/2015 07:43 PM - Roland Schulz

I think schema validation is a requirement for at least some of the files. It isn't ready in rapidjson yet .

#20 - 10/27/2015 08:27 AM - David van der Spoel

Once again it seems discussion runs into the sand.
Could an option be to keep using xml for the testing of code (not normally performed by users and not strictly necessary on all platforms) and to use something else (e.g. json) for user files?

Implementing yet another package requires IMHO that it is shipped with GROMACS. rapidjson could fit the bill although the license demands the software is used for Good not Evil, which is difficult to guarantee (I am pretty confident that some GROMACS users works on explosives with military objectives for instance).

#21 - 10/27/2015 10:08 AM - Erik Lindahl

The license thing has already been resolved on debian-legal; apparently that clause only applies to the small separate JSON_checker program (400 lines), while rapidjson itself is strict MIT.

#22 - 10/27/2015 10:33 AM - David van der Spoel

OK, fine. How about using both xml and json?

#23 - 10/27/2015 11:58 AM - David van der Spoel

For testing I added rapidjson to the src/external directory, it is no problem including it, but it generates a ton of warnings from doxygen. I guess we can suppress those, right?

#24 - 10/27/2015 12:09 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1842.
Uploader: David van der Spoel (davidvanderspoel@gmail.com)
Change-Id: I248f048556f601ebcecdbe405b7785d92c9b6e08
Gerrit URL: https://gerrit.gromacs.org/5247

#25 - 10/27/2015 07:37 PM - Roland Schulz

I don't think we should use both.

Teemu said that the non-xslt part is relative easy to convert. If we are OK with converting

```
<Bool Name="Dynamic">false</Bool>
```

into something like

```
      "Bool": {
            "-Name": "Dynamic",
            "#text": "true"
          }
```

Then the json->xml->xslt would be no problem and converting the pretty print for the test is easy. If we convert xml->json in a way that doesn't preserve what was attribute and text than we couldn't keep using the xslt as is. Would we want a format for the referencedata in json which is reversible or something which is a bit more natural for a format which doesn't have attributes or text? Such as

```
  "Bool": {
            "Dynamic": "true"
          }
```

#26 - 11/06/2015 05:51 AM - Teemu Murtola

Neither of those probably work, since there might be several Bool elements within the same parent element, and that is not valid JSON. And they both make multiple lines for each entry, which likely will hurt readability.

Rather, I think we should just convert the example to

```
"Dynamic" : true
```

(or to a string "true"), and then have a separate file (or section within the file) that specifies the types for the elements as

```
"Dynamic" : "Bool"
```

It is possible to share these type definitions between multiple tests as well, probably with not too much effort.

The thing that requires a bit more thinking is how to represent sequences. Best approach might involve some changes also in the tests that test sequences and in the API they use for this, but this shouldn't still be too hard.

#27 - 11/08/2015 09:35 AM - David van der Spoel

Now that patch https://gerrit.gromacs.org/#/c/5247/18 seems ready to be included (if we decide on using RapidJSON), we should consider the next step.

Teemu proposed to have a wrapper around the library, to hide its details to the user. This could be a class implementing a tree structure that is built up in memory by the user and then written using the JSON library. However, since it is possible that files/memory use can become large it will be necessary to have the possibility to write blocks as well.

I'm still not convinced JSON will fit our needs better than XML, but we need a decision such that we can get on with coding.

#28 - 11/08/2015 06:32 PM - Roland Schulz

What's the advantage of the wrapper? Hopefully we don't plan to change it again anytime soon. Note that the wrapper would also add maintenance and complexity. So it might only be worthwhile after changing it TWO more times.

#29 - 11/08/2015 06:38 PM - Erik Lindahl

We don't want a wrapper just for rapidJSON, but a higher-level module that represents data being read/written so we don't start to have hundreds of low-level JSON calls all over the code.

#30 - 11/08/2015 09:11 PM - Teemu Murtola

Roland Schulz wrote:

> What's the advantage of the wrapper? Hopefully we don't plan to change it again anytime soon.

That gives us a single point where we can adapt the behavior of the external library to our needs when it comes to, e.g., error handling, or the set of operations we want to support. And I haven't really seen a very thorough evaluation of different JSON libraries that led to picking just RapidJSON... Who says that we do not want to change to some other implementation in the future? Or to YAML?

> Note that the wrapper would also add maintenance and complexity. So it might only be worthwhile after changing it TWO more times.

That's only the case if you plan to changing it twice without writing any code that actually uses it before the changes... The point of the wrapper is to encapsulate the use of the external library; it doesn't really reduce maintenance or complexity if you instead have every single user take care of, e.g., similar error handling tasks. Even if you have just a few file formats using it, if those all are written directly to operate on RapidJSON data structures throughout, there's easily an order or magnitude more code specific to RapidJSON than if they were using a common wrapper. And then suddenly the wrapper is worthfile even if you just change the external library 0.1 times...

No matter which approach we pick, whether it is XML or JSON, and whether it is RapidJSON or something else, I think what should happen is that we should take an existing file format (or a few), and convert those to use the new approach. Just writing a wrapper without any code that is going to use it is not going to be useful (and with high certainty does not actually solve the actual problems).

I would take, for example, the atomprop.cpp functionality, which is relatively small and self-contained, and has various types of structured data. I think this would help the discussion more than starting to use the new library only in some new feature that gets introduced in a 10k-line commit that takes ages to review and get agreement on all other details irrelevant to this discussion.

#31 - 07/28/2016 09:27 AM - Teemu Murtola

- Related to Task #2017: Modularize simulation option storage and reading from mdp files added

#32 - 09/30/2016 09:56 AM - David van der Spoel

Teemu made a couple of patches to store key-value pairs in the code, e.g. https://gerrit.gromacs.org/#/c/6057/. In order to move forward with the JSON patch I guess it would be good to implement a KeyValueTreeBuilder that reads and writes JSON files. Please correct me if I'm wrong or if there are better ways. Then once that is in place we can convert e.g. the atommass.dat and friends to JSON and replace gmx_atomprop_t. Comments?

#33 - 09/30/2016 11:27 AM - Dawei Si

I just had a look at the code around KeyValueTree and atomprop. The current implementation of `KeyValueTreeBuilder` seems to be useful. Have read the discussion above, I suggest why not consider using HDF5 for very large datafiles? It is efficient and can be partially read/written, although I guess this can also bring annoying compatibility issues.

Starting with `atommass.dat` seems reasonable. Maybe I could add the wrapper functions of RapidJSON? BTW, if we use JSON, RapidJSON should be the best choice (see https://github.com/miloyip/nativejson-benchmark)

And there's another very small issue: many RapidJSON header files you included have windows-style `\r`s which make them not neat. (I'll fix it)

#34 - 09/30/2016 12:28 PM - David van der Spoel

Just for clarity, this kind of files is not intended for big binary data, just for relatively small text files. FYI, we have discussed HDF5 previously for the big trajectory files we have but decided against it.

#35 - 10/04/2016 04:04 PM - Dawei Si

I want to discuss an issue about the current `KeyValueTreeBuilder`. In my opinion, the return type of `KeyValueTreeBuilder::build()` should be `KeyValueTreeValue` instead of `KeyValueTreeObject`, which means the root of a `KeyValueTree` doesn't need to be a key-value mapping object, but can also be an array (and even in basic type such as string or numbers).

Thus when reading and writing data with a file, we don't have to force the data to be a key-value mapping object, and then we can directly store arrays in the file. Since `KeyValueTreeValue` has methods like `asArray()` and `asObject`, I think it's the perfect choice to be the root of data.

#36 - 10/20/2016 09:25 AM - Erik Lindahl

We talked a bit about JSON at the hackaton, and one of the outcomes was that we realized the central part isn't the file format(s), but well-defined and stable APIs to handle various type of data at the conceptual level rather than low-level parsing of values.

Once that is in place, it's a close-to-trivial exercise to just read/write the data from a particular type of format.

#37 - 01/05/2017 11:48 AM - Gerrit Code Review Bot

Gerrit received a related patchset '25' for Issue #1842.
Uploader: David van der Spoel (davidvanderspoel@gmail.com)
Change-Id: gromacs~master~I248f048556f601ebcecdbe405b7785d92c9b6e08
Gerrit URL: https://gerrit.gromacs.org/5247

#38 - 01/15/2018 03:51 PM - Aleksei Iupinov

- Related to Task #2383: Add JSON interface to write and read files added

#39 - 01/15/2018 04:24 PM - Erik Lindahl

Mark and I had another chat about JSON and our best strategies for it, partly based on recent development in JSON libraries.

One approach is of course the one we have talked about above, i.e. having separate JSON functionality stuff in the IO module. However, the drawback with that is that we might end up with the IO module having to know a lot about the structure of our data, and at the end there will still have to be one implementation file for each major structure we want to write, as well as some sort of low-level interface.

However, another option could be to let each (relevant) object have serialization methods. At least the "JSON for Modern C++" library by Niels Lohmann (https://nlohmann.github.io/json/) hides a lot of the complexity so we can just serialize the basic C++11 containers directly. That might help avoid a lot of bugs, and by having the serialization as part of the object we can easily catch bugs by testing serializetion/deserialization; if we forget to save/restore any value we should hopefully notice.

The drawback is of course that we might end up with JSON in lots of places. If we do not like that we could imagine writing our own wrapper, but still largely subscribing to the C++11 and assignment idea.

Still, overall I fancy the idea that serialization is a method for each object, rather than a separate third-party functionality dealing with lots of large/complex class hierarchies.

Opinions?

#40 - 01/15/2018 04:41 PM - Paul Bauer

I also started looking into the library you mentioned, as it is pure C++ and has already test coverage and everything, with hiding the more complex side being an added bonus. Reading up now on how to use it for some test cases.
I don't think GROMACS should have its own wrapper, because this would just mean that there will be all possible kinds of bugs introduced by doing things from scratch.

#41 - 01/18/2018 05:12 PM - Roland Schulz

I think this is a much better solution. I also agree we shouldn't have a wrapper. Note that GCC 4.8 isn't officially supported by the library. But hopefully we anyhow bump the required GCC version so that this isn't an issue.

Files

xml_json.png	101 KB	10/12/2015	Erik Lindahl
json_xml.png	14.1 KB	10/12/2015	Roland Schulz