Felix' TODO list (July 14), Klaus' remarks (Aug 13)

The requests below were implemented, mostly for boost::multi_array. Please see example/dataset.cpp for an overview on the current capabilities of the code. Some unit test cases are still missing, though.

* implement class h5xx::dataset, in analogy to h5xx::attribute

  As a difference, multiple writes to different hyperslabs of the same dataset
  can occur (incremental write). Hence, creation and writing have to be
  separate functions.

Creation and write routines are implemented. The write routines do not create, by default.

  Suggestion: creation occurs in the constructor (or in a free function?),
  writing is separate free function and will eventually support
  hyperslab addressing for incremental and parallel I/O.

Creation routines are implemented as free functions (see scalar.hpp, boost_multi_array.hpp). Moreover, a generic create_dataset free function exists which takes h5xx::dataspace and h5xx::datatype arguments. In addition, constructors to create datasets are implemented. Simple hyperslab support (offset, count) is implemented for boost::multi_array. Read and write functions take dataspace objects which must be manipulated using dataspace.select_hyperslab(). This should be simplified further, probably using and overloaded [] operator which takes hyperslab indices.

  Example:

```
    // create dataset via constructor
    h5xx::dataset dataset(group, name, datatype, dataspace, policies); // variant of the constructor
```
Implemented.

```
    // alternative: pass data variable to derive datatype and dataspace,
    // template specialisation is simpler if it is a free function
    template <typename T>
    h5xx::dataset create_dataset(group, name, data);
```
Implemented for simple scalars and boost:multi_array.

```
    // write to dataset (free function)
    template <typename T>
    h5xx::write<T>(h5xx::dataset const& dataset, T const& data, hyperslab_index);
```
Implemented for simple scalars and boost:multi_array.

```
    // read from dataset,
    // passing data as argument gives automatically access to the type T
    template <typename T>
    h5xx::read<T>(h5xx::dataset const& dataset, T& data, hyperslab_index);
```
Implemented for simple scalars and boost:multi_array.

```
* implement policies for h5xx::dataset constructor (see StringPolicy in attribute/scalar.hpp):
    - storage_layout (compact, contiguous, chunked)
    - filters (deflate, shuffle), parameter for chunked layout
    - fill_value

    - example:

    auto storage_policy = policy::storage::chunked(boost::array const& shape, policy::filter::deflate(6));
    h5xx::dataset ds(..., storage_policy, fill_value); // create dataset via constructor
    write(ds, data);
```
Implemented, please see policy.hpp and example/dataset.cpp. I am sure there are suggestions for improvement.

```
* implement hyperslab (policy?) for reading/writing h5xx::dataset, can we
  introduce a slicing notation as in Python?
```
See previous comment.

```
* implement class h5xx::datatype, we have already h5xx::ctype<T>
  (removing any explicit calls to H5T*, look at output of `grep -R H5T h5xx/`)
```
Basic class is implemented, support for boost::multiarray.

```
* make support of Boost C++ types optional to relax build requirements
```
std::vector can be used instead of boost::arrays whenever „auxiliary“ arrays are needed. As an example, dataset.select_hyperslab was implemented in both variants.