Hierarchical dynamic containers for fusion data

David Fridrich    Jakub Urban

Institute of Plasma Physics of the CAS, Czech Republic
Motivation

- Scientific data can be organized into hierarchical tree structures.
- One can find several examples:
  - Standard data exchange formats: XML, JSON, HDF5,
  - EUROfusion Integrated Modelling (EU-IM),
  - Integrated Modelling & Analysis Suite (IMAS) datatypes, ...
  - ...similarly most experimental data within fusion community.
- Either for integrated modelling or data processing one needs to exchange data
  - among multiple threads, processes or even cluster nodes,
  - as rapidly as possible \( \Rightarrow \) to avoid unnecessary copying,
  - minimizing memory footprint.
Create a data exchange library that will

- be flexible and runtime oriented ⇒ no recompilation of entire project due to single change in data tree,
- have efficient and uncomplicated interfaces for the most used programming languages,
- allow straightforward (de)serialization into commonly used data formats,
- scale from small single user tasks to large projects,
- provide efficient inter-process communication on multi-core systems with private, shared or distributed memory.
Project goals

- Single API: Specify data system
- Simulation code
- Rapid, shared/private memory communication
- Data exchange within cluster nodes
- Automatic conversions

DATA SYSTEMS:
- JET
- ITM/WPCD
- TRANSP
- IMAS
- COMPASS
- ...

Fridrich, Urban: HDC for fusion data
Hierarchical Dynamic Containers

- C++ library for rapid exchange of scientific data,
- Provides bindings for C, Fortran and Python languages (planned support for Matlab, Java, ...)
- Supports different storage backends via plugin system.
- Depending on selected storage plugin, it can provide requested features:
  - data persistence, in-core operation (zero-copy), ...
- Available under the MIT licence.
We want to build on well known abstractions:
- tree structure for **frontend** – well established in scientific community,
- key-value store for **backend** – several existing solutions, easy to select specific one for given purpose.

Multiple existing key-value stores with different properties can be used:
- `std::unordered_map` for private memory,
- Yahoo MDBM for inter-process communication,
- memcached / redis for data sharing among multiple cluster nodes.

Supported datatypes:
- similar to numpy: scalars & arrays, string, map, list, empty node
Implementation

- Each node of tree is represented by single entry in key-value store.
- Key is fixed size UUID.
- Value is contiguous buffer consisting of header and data.
- Header contains following metadata:
  - size of buffer, data type, number of dimensions, array shape,
  - additional flags (access policy, locking information, data endianity, array ordering, etc)
- Tree hierarchy is mapped using child references ⇒ non-terminal nodes (= maps and lists) contain references to their children.
- Maps and lists use boost::multi_index ordered hash map embedded into in boost::managed_external_buffer.
- Arrays of numerical types are exposed using view-based libraries:
  - andres::marray for C++
  - numpy.array for Python
Basic usage

Python

```python
import numpy as np
from pyhdc2 import HDC

# create tree
tree = HDC()

# prepare some data
data = np.array([7, 2, 3, 4])

# store some data
tree["some/path"] = data

# get data back
data2 = np.asarray(tree["some/path"])

# get subtree
subtree = tree["some/path"]
```

C

```c
#include "hdc_c.h"

// create tree
struct hdc_t* t = hdc_new_empty();

// Prepare some data
int32_t arr[4] = {7, 2, 3, 4};
size_t shape[1];
shape[0] = 4;
// Add data to a single node
struct hdc_t* data = hdc_new_empty();
hdc_set_int32(data, 1, shape, (void*)arr);  // Get data back from tree
int32_t* arr2 = hdc_as_int32_1d(node);
// get subtree
struct hdc_t* subtree = hdc_get(data, "some/path");
```
current status & future plans

**Current status**
- technical preview,
- concepts work well, but limited data types support,
- supported (de-)serialization from/to HDF5 and JSON,
- testing IMAS integration,
- planned as a part of EEG-2017/11 grant.

**Future plans**
- locking mechanism,
- garbage collector,
- more datatypes support,
- bindings to more languages,
- tokamak databases access,
- toolkit for common data tasks:
  - various data sources access, data conversion, parsing, ... 

**How to obtain**
git clone https://bitbucket.org/compass-tokamak/hdc.git
cd hdc
git checkout next
and see README.md for details...