Conduit

Scientific Data Exchange Library for HPC Simulations

Cyrus Harrison, Brian Ryujin, Adam Kunen

Lawrence Livermore National Laboratory

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

LLNL-PRES-666591

Data coupling is a key aspect in simulation software design and user workflows.

Today:

I/O libraries have evolved into defacto interfaces between:

- Simulation Components
- Simulations and Pre- and Post-processing tools:
  - meshing, visualization, analysis, etc

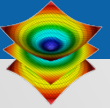
I/O is acceptable as a coarse-grain and low-frequency data coupling solution.

Data coupling is a key aspect in simulation software design and user workflows.

The Future:

- Increased CS emphasis on modular physics package design
- In-situ APIs for components that are usually I/O isolated as Pre- and Post-processing

To achieve fine-grain and high-frequency data coupling we need tools to help with in-core data exchange.



Supporting in-core data exchange throughout the simulation eco-system is quite different from trivial single component use cases.

Key Requirements:

- A description mechanism for numeric primitives:
  - Scalars, ragged arrays, etc with explicit precision
- Mixed memory ownership semantics:
  - Enables Zero-copy where feasible
  - Plays friendly with existing data structures
- Enable higher level conventions:
  - Hierarchical context
  - Human readable descriptions

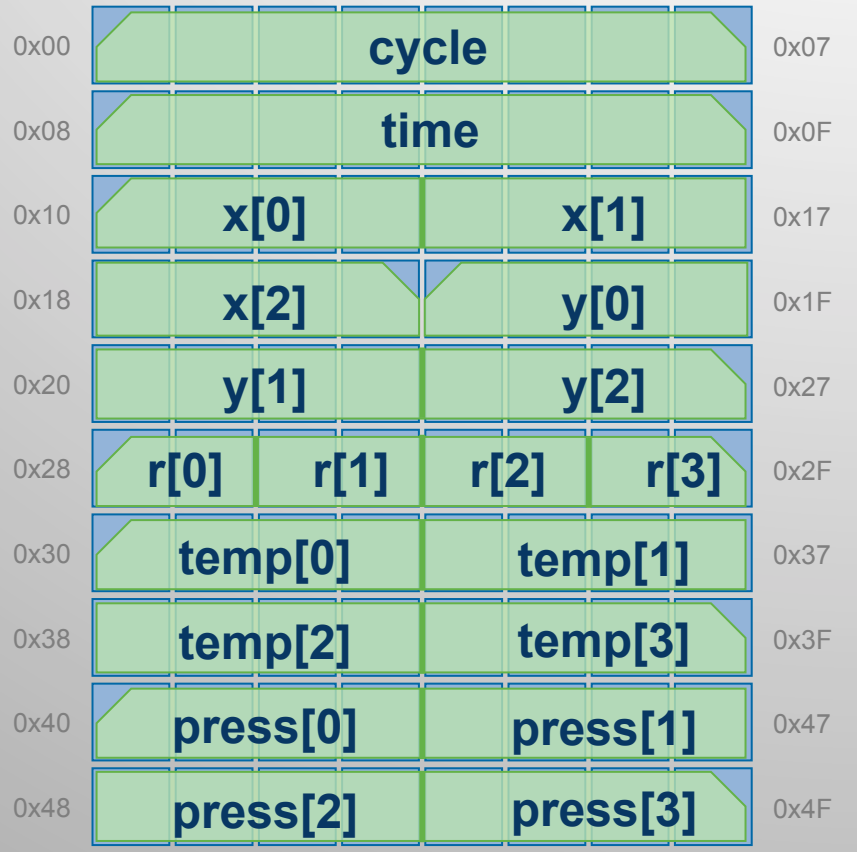
Conduit is a new open source development effort at LLNL aimed at simplifying in-core data exchange.

Conduit provides:

- A flexible way to ***describe*** complex data:
  - A JSON-based schema for describing the layout of hierarchical in-core data.
- A sane API to ***access*** complex data:
  - A dynamic API for rapid construction and consumption of hierarchical data in C++, C, Python, and FORTRAN.

Our goal is to create a small library that can be used for data description and exchange in HPC codes.

Example: Describing in-core data using a Conduit Schema.



Data (80 bytes)

```
{
 "cycle": "uint64",
 "time": "float64",
 "coords":
 {
  "x": {"dtype":"float32", "length": 3},
  "y": {"dtype":"float32", "length": 3}
 },
 "fields":
 {
  "region": {"dtype":"uint16", "length": 4},
  "temp": {"dtype":"float32", "length": 4},
  "pressure": {"dtype":"float32", "length": 4}
 }
}
```

Conduit JSON Schema

Example: Accessing in-core data via a Conduit Schema using C++.

```
// start with a data pointer and our example schema
void   *data_ptr    = ...
string  json_schema = ...

// construct a Node from the schema and data
Node n(json_schema,data_ptr);

// print the cycle and time values
cout << "cycle = " << n["cycle"].as_uint64() << endl;
cout << "time  = " << n["time"].as_float64() << endl;
```

```
cycle = 100
time  = 2.8
```

Example: Accessing in-core data via a Conduit Schema using C++.

```
// print the x coordinate values
float32 *x_coords = n["coords/x"].as_float32_ptr();
cout << "x[0]  = " << x_coords[0] << endl;
cout << "x[1]  = " << x_coords[1] << endl;
cout << "x[2]  = " << x_coords[2] << endl;
```

```
x[0]  = 0
x[1]  = 0.25
x[2]  = 1
```

Example: Accessing in-core data via a Conduit Schema using C++.

```
// access the region field and print its first value
Node &fields = n["fields"];
uint16 *reg_ptr = fields["region"].as_uint16_ptr();
cout << "r[0]  = " << reg_ptr[0] << endl;
```

```
r[0]  = 0
```

Example: Accessing in-core data via a Conduit Schema using C++.

```
// print entire node in a human readable fashion
n.print();
```

```
{
  "cycle": 100,
  "time": 2.8,
  "coords":
  {
    "x": [0, 0.25, 1],
    "y": [0, 0.5, 1]
  },
  "fields":
  {
    "region": [0, 1, 0, 1],
    "temp": [1, 2, 3, 4],
    "pressure": [0.1, 0.3, 0.5, 0.7]
  }
}
```

Example: Dynamic object construction with Conduit in C++.

```
// create a Node instance
Node n;
// add cycle and time values
uint64  cyc  = 100;
float64 time = 2.8;
n["cycle"] = cyc;
n["time"]  = time;
n.print();
```

```
{
  "cycle": 100,
  "time": 2.8
}
```

Example: Dynamic object construction with Conduit in C++.

```
// add x and y coords
float32 x[3] = {0.0,0.25,1.0};
float32 y[3] = {0.0,0.5,1.0};

n["coords/x"].set(x,3);
n["coords/y"].set(y,3);
n.print();
```

```
{
  "cycle": 100,
  "time": 2.8,
  "coords":
  {
    "x": [0, 0.25, 1],
    "y": [0, 0.5, 1]
  }
}
```

Example: Dynamic object construction with Conduit in C++.

```
// add fields
uint16  r[4] = {0,1,0,1};
float32 t[4] = {1.0,2.0,3.0,4.0};
float32 p[4] = {0.1,0.3,0.5,0.7};

n["fields/region"].set(r,4);
n["fields/temp"].set(t,4);
n["fields/pressure"].set(p,4);
n.print();
```

```
{
  "cycle": 100,
  "time": 2.8,
  "coords":
  {
    "x": [0, 0.25, 1],
    "y": [0, 0.5, 1]
  },
  "fields":
  {
    "region": [0, 1, 0, 1],
    "temp": [1, 2, 3, 4],
    "pressure": [0.1, 0.3, 0.5, 0.7]
  }
}
```

The heart of Conduit is a hierarchical variant type named *Node*.

A *Node* acts as one of following basic types:

- Object
  - An ordered associative array mapping names to children.
- List
  - An ordered list of unnamed children.
- Leaf (Scalar or Array of bitwidth-specified primitives)
  - Signed Integers: int8, int16, int32, int64
  - Unsigned Integers: uint8, uint16, uint32, uint64
  - Floating Point Numbers: float32, float64
  - Strings: char8_str

Conduit was designed with software engineering eco-system logistics in mind.

- Completely runtime focused
  - Avoids incompatible (or unsharable) code-generation solutions.
- Language agnostic
  - C++ API which underpins developing Python, C, and Fortran APIs.
  - JSON is used for data layout
- Data description as a core capability
  - Does not require repacking
  - Helps build serialization, I/O, and messaging features.

Philosophy: Share data without massive code infrastructure.

Conclusion

- **Conduit** can help ease in-core data exchange in our simulation codes and are working eagerly to test this hypothesis.

- **Conduit** is released under a BSD-Style License.

- Contact Info:

Cyrus Harrison (cyrush@llnl.gov)

Brian Ryujin (ryujin1@llnl.gov)

Adam Kunen (kunen1@llnl.gov)