**Configuration validation in the art event-processing framework**
Kyle J. Knoepfel, Fermi National Accelerator Laboratory

**Introduction**

The art event-processing framework loads at run-time a set of modules to create workflows that serve in data acquisition, simulation generation, reconstruction algorithm execution, and physics analysis. art processes are configured by a collection of user-specified parameters defined using the FHiCL language. To aid users in properly configuring their processes, a configuration retrieval, description, and validation suite has been implemented that:

- has a single point of maintenance for users,
- can represent in C++ an arbitrary FHiCL structure, and
- can be enabled at the user level without breaking existing workflows and C++ source code.

**FHiCL language**

The Fermilab Hierarchical Configuration Language is the language used for configuring art processes. Configuration parameter names and their associated values are classified in three ways:

- **Atom**: a named value with no underlying structure.
- **Sequence**: a named list of unnamed values.
- **Table**: a named collection of name-value pairs.

The above example is a typical module configuration. Note that nested tables and sequences (and tables in sequences) are allowed, leading to configurations with arbitrary depth.

**Configuration representation in C++**

If a user wants to validate the above configuration in his/her module, the following would be specified in the C++ source code:

```cpp
struct Config {
  Atom<bool> verbose (Name("verbose"));
  Sequence<int> particleIDs (Name("particleIDs"));

  struct G4Settings {
    Atom<string> shape (Name("shape"), "sphere");
    Atom<double> radius (Name("radius"),
                           Comment("Units are mm.")));
    Table<G4Settings> g4Settings (Name("g4Settings"));
  } g4Settings;
};
```

The Atom, Sequence, and Table class templates receive a template argument that specifies the type to which the FHiCL parameter should be converted within the C++ code.

The validation system supports defaults in source code (e.g. the shape parameter), as well as comments to be printed out when the description is requested (e.g. the radius parameter).

**Module description and validation**

For a module that supports the configuration above (e.g. G4Module), a description similar to the one at the right is provided by art.

Suppose a user were to mis-specify shape as ‘Shape’, an error similar to the following would be emitted:

```cpp
Any parameters prefaced with ‘#’ are optional.
Unsupported parameters:
+ g4Settings.Shape
```

**C++ implementation aspects**

The implementation of the suite relies heavily on modern C++ features include:

- variadic templates for representing heterogeneous sequences,
- lambda expressions for configuration tree-walking and conditional configuration based on the value of a previously validated parameter,
- automatic type deduction, etc.

Adoption of such C++ techniques provides a type-safe suite, moving error detection, when possible, to the compile-time stage instead of the run-time stage.

**Additional features**

The design of the suite was informed from interactions with art users. Based on those discussions, a large number of additional features have been included in the suite. Additional parameter types include:

- **optional parameters** – where it is permitted to omit supported parameters without specifying a default in source code,
- **delegated parameters** – where the parameter itself must be present, but its value (atom, sequence, or table) is unspecified,
- **conversion parameters** – where a configuration sequence can be converted directly to a user-specified type, without having to retrieve “by hand” individual sequence elements and convert them to the relevant type.

**Deployment in art and its experiments**

User feedback regarding the suite has been positive. Almost all art-provided facilities enable configuration validation and description. Individual experiments that use art are adopting the suite in their own code according to their own needs.