fhicl-cpp - Feature #23669

Provide a FHiCL type to read an object arbitrary type from a parameter table

11/29/2019 05:42 PM - Gianluca Petrillo

Status:	Closed	**Start date:**	11/29/2019
Priority:	Normal	**Due date:**	
Assignee:	Kyle Knoepfel	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	3.06.00	**Spent time:**	3.00 hours

Description

FHiCL currently provides fhicl::TupleAs<> allowing to create an arbitrary type from a sequence: [1.0, 2.0, 1.0, blue] may become a ColorPoint.
For more complex data structures the sequence fits tight, and the full power of a Table may be desired: { point: [1.0, 2.0, 1.0] color: blue }.
This allows a better control on the single components and a more precise documentation. It also gives more flexibility (e.g. a default colour value).
I would like an entity fhicl::TableAs (and its fhicl::OptionalTableAs sibling) to be introduced, which accepts some form of function to convert the configuration structure in the table into a single object.
A rough sketch of how the usage I foresee:

```
struct ColorPoint {
  Point_t point;
  Color_t color;
};

struct ColorPointConfig {
  fhicl::TupleAs<Point_t(double, double, double)> point
    { fhicl::Name{ "Point" }, fhicl::Comment{ "position: [ x, y, z ] in global coordinates [cm]"
 } };
  fhicl::Atom<Color_t> color
    { fhicl::Name{ "color" }, fhicl::Comment{ "color of the point" }, makeColor("black")
/* default */ };

  ColorPoint operator() () const;
}; // struct ColorPointConfig

struct Config {

  fhicl::TableAs<ColorPointConfig, &ColorPointConfig::operator()> point {
    fhicl::Name{ "point" },
    fhicl::Comment{ "reference point (including location and color)" },
    ColorPoint{ Point_t{}, makeColor{ "black" } }
    };

}; // struct Config

void parse(art::EDProducer::Table<Config> const& config) {
  auto pc = config().point(); // this is a ColorPoint object
  // ...
}
```

(please ignore any "alternative C++" syntax that I may have unintentionally introduced)
The intention is that the conversion function, second template parameter of TableAs, may be any function taking the table structure as argument.
A more restrictive pattern might instead require the Config struct be convertible to the target type, now with

```
struct ColorPointConfig {
  fhicl::TupleAs<Point_t(double, double, double)> point
    { fhicl::Name{ "Point" }, fhicl::Comment{ "position: [ x, y, z ] in global coordinates [cm]"
```

```
 } };
  fhicl::Atom<Color_t> color
    { fhicl::Name{ "color" }, fhicl::Comment{ "color of the point" }, makeColor("black")
/* default */ };

  explicit operator ColorPoint() const; // now with 56% more conversion!
}; // struct ColorPointConfig

struct Config {

  fhicl::TableAs<ColorPointConfig, ColorPoint> point {
// ... and we still need to specify the target type but not any more how to get it
    fhicl::Name{ "point" },
    fhicl::Comment{ "reference point (including location and color)" },
    ColorPoint{ Point_t{}, makeColor{ "black" } }
    };

}; // struct Config
```

or require an operator() in the fashion of the first example, which might make its implementation a more straightforward extension of fhicl::Table.

History

#1 - 12/02/2019 10:37 AM - Kyle Knoepfel

- Status changed from New to Under Discussion

This will take some discussion, Gianluca. Happy to set up a discussion with you.

#2 - 01/09/2020 01:50 PM - Kyle Knoepfel

- Status changed from Under Discussion to Accepted

Upon discussion with Gianluca, the proposal is to provide a class template with the signature fhicl::TableAs with the following behavior. Consider the arbitrary class

```
namespace a {
  class A {...};
}
```

If a configuration class and conversion function are specified in namespace 'b' (e.g.)

```
namespace b {
  struct ConfigForA {
    fhicl::Atom<bool> flag{fhicl::Name{"flag"}};
    ...
  };
  a::A convert(ConfigForA const& config) { return a::A{...}; }
}
```

then calling the function 'fhicl::TableAs<a::A, b::ConfigForA>::operator()' will return an object of type a::A according to the conversion function specified in namespace 'b'. Specifically:

- a::A must specify the desired C++ type
- b::ConfigForA must specify the configuration struct
- The conversion function must reside in the same namespace as the configuration struct, with the name 'convert' that receives the configuration struct and returns the desired C++ type.

#3 - 02/17/2020 12:26 PM - Gianluca Petrillo

I should add that I would like this new entity to be supported in sequences too: fhicl::Sequence<fhicl::TableAs<T, Conv>> yielding a std::vector<T> (if I recollect right, fhicl::Table are supported in that way).

#4 - 06/09/2020 07:54 AM - Kyle Knoepfel

- Target version set to 3.06.00

#5 - 06/10/2020 04:32 PM - Kyle Knoepfel

- % Done changed from 0 to 100

- Assignee set to Kyle Knoepfel

- Status changed from Accepted to Resolved

The fhicl::TableAs and fhicl::OptionalTableAs class templates have been implemented with commit fhicl-cpp:8d8f81f, supporting automatic conversions from FHiCL tables to user-defined C++ types. Sequences of fhicl::TableAs template instantiations are also supported.

#6 - 07/14/2020 10:37 AM - Kyle Knoepfel

- Status changed from Resolved to Closed