

fhicl-cpp - Feature #9079

Feature # 9078 (Closed): ParameterSet validation enhancements

Conditional configuration for fhiclcpp types

06/09/2015 09:08 AM - Kyle Knoepfel

Status:	Closed	Start date:	06/09/2015
Priority:	Normal	Due date:	
Assignee:	Kyle Knoepfel	% Done:	100%
Category:		Estimated time:	16.00 hours
Target version:	1.17.05	Spent time:	9.00 hours
Description			
<p>Mu2e and NOvA have expressed interest in supporting conditional configuration, as described in #8773-3. There are a couple different ways this feature could be implemented. The first is to allow some parameters to be unvalidated:</p>			
<pre>Atom<std::string> value1 { Key("value1"), mode::Unchecked }; Atom<std::string> value2 { Key("value2"), mode::Unchecked };</pre>			
<p>which would allow users to access type but not require that it be present in the FHiCL configuration file. The code would throw if a retrieval was requested and a value was not provided in the FHiCL file. This option would be a placeholder for something more complete, like:</p>			
<pre>Atom<int> type { Key("type") }; Atom<std::string> value1 { Key("value1"), If(typeIs, 1) }; Atom<std::string> value2 { Key("value2"), If(typeIs, 2) };</pre>			
<p>where typeIs is some user-provided function (pointer) that queries the value of type. The latter option, while most helpful for the user, would require that value checking would need to take place for certain variables before key-checking (current behavior) could resume. One would then need to specify the behavior (and possibly new syntax) for cases when the argument following typeIs is not an allowed value for type.</p>			
Related issues:			
Blocks fhicl-cpp - Feature #10820: Parameter validation based on limits		Closed	11/05/2015

History

#1 - 06/09/2015 09:08 AM - Kyle Knoepfel

- Related to Feature #8773: Create new types in FHiCL that automatically register allowed parameters. added

#2 - 06/09/2015 09:09 AM - Kyle Knoepfel

- Related to deleted (Feature #8773: Create new types in FHiCL that automatically register allowed parameters.)

#3 - 06/09/2015 09:09 AM - Kyle Knoepfel

- Parent task set to #9078

#4 - 06/15/2015 11:24 AM - Christopher Green

- Category set to Infrastructure

- Status changed from New to Feedback

- Assignee set to Kyle Knoepfel

- Target version set to 2.01.00

Analysis is required for a reasonable time estimate.

#5 - 07/02/2015 02:45 PM - Kyle Knoepfel

- Subject changed from Conditional configuration for new FHiCL c++ types to Conditional configuration for fhiclcpp types

#6 - 08/26/2015 11:37 AM - Kyle Knoepfel

- Project changed from art to fhicl-cpp

- Category deleted (Infrastructure)

#7 - 11/09/2015 11:49 AM - Christopher Green

- Blocks Feature #10820: Parameter validation based on limits added

#8 - 12/09/2015 11:50 AM - Kyle Knoepfel

- Status changed from Feedback to Assigned

- Estimated time set to 16.00 h

#9 - 12/10/2015 09:22 AM - Kyle Knoepfel

- % Done changed from 0 to 40

#10 - 12/10/2015 11:02 AM - Kyle Knoepfel

- % Done changed from 40 to 80

#11 - 12/10/2015 01:17 PM - Kyle Knoepfel

- Status changed from Assigned to Resolved

- % Done changed from 80 to 100

Implemented with commit [fhicl-cpp:3c8cee0](#). Documentation will be forthcoming. However, as a preview, the conditional configuration for a given module is specified using an `std::function<bool()>` object. For example, consider the following configuration:

```
struct BoxParams { Atom<>... };
struct SphereParams { Atom<>... };

struct Config {
  Atom<int> someNumber { Name("someNumber") };
  Atom<bool> flag { Name("flag") };
  Sequence<int> list { Name("list") };
  Atom<string> shape { Name("shape") };

  Table<BoxParams> boxParams { Name("boxParams"),
                              Comment("Use if shape = box."),
                              [this]() { return shape() == "box"; } };

  bool maybeSphere() const
  {
    return someNumber() == 10 &&
           flag() &&
           list().size() == 17 &&
           shape() == "sphere";
  }

  Table<SphereParams> sphereParams { Name("sphereParams"),
                                     Comment("Use if the member function 'maybeSphere' returns true."),
                                     fhicl::use_if(this, &Config::maybeSphere) };
};
```

For simple conditions, the lambda closure (as used for `boxParams`), is the simplest way to create the `std::function<bool()>` object. For more complicated conditions (as in the horribly contrived `sphereParams` case), one can use the `fhicl::use_if` and `fhicl::use_unless` helpers to create the `std::function<bool()>` object for you. In both cases, the predicate function (either the anonymous lambda or `maybeSphere`) must return a `bool` and it can receive no arguments.

Full documentation is [here](#).

#12 - 12/17/2015 01:43 PM - Kyle Knoepfel

- Target version changed from 2.01.00 to 1.17.05

#13 - 12/23/2015 09:00 AM - Kyle Knoepfel

- Status changed from Resolved to Closed