art and `ParameterSet` validation:

Technology preview

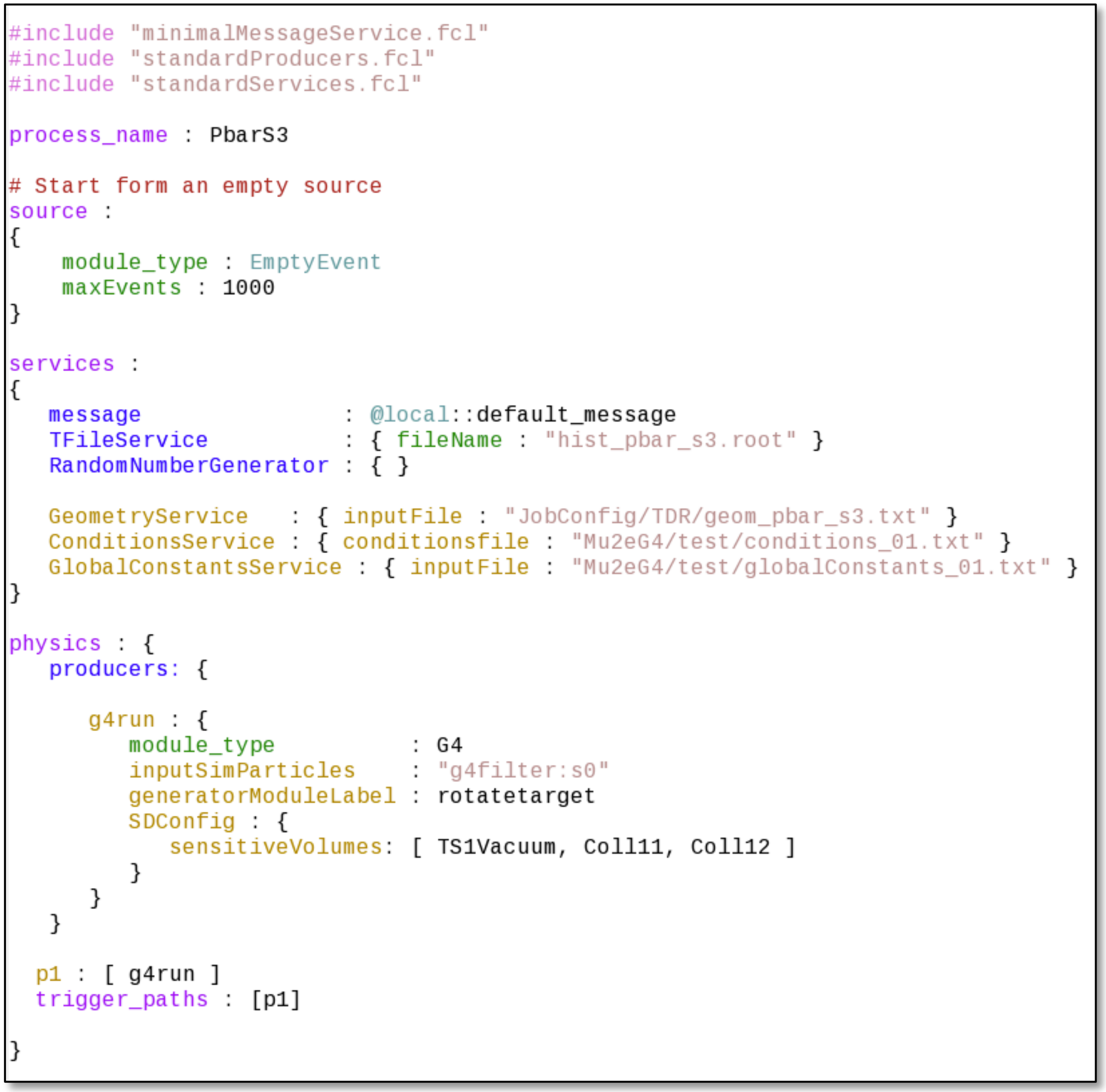
Kyle J. Knoepfel

June 18, 2015

Fermi National Accelerator laboratory

FHiCL and *art*

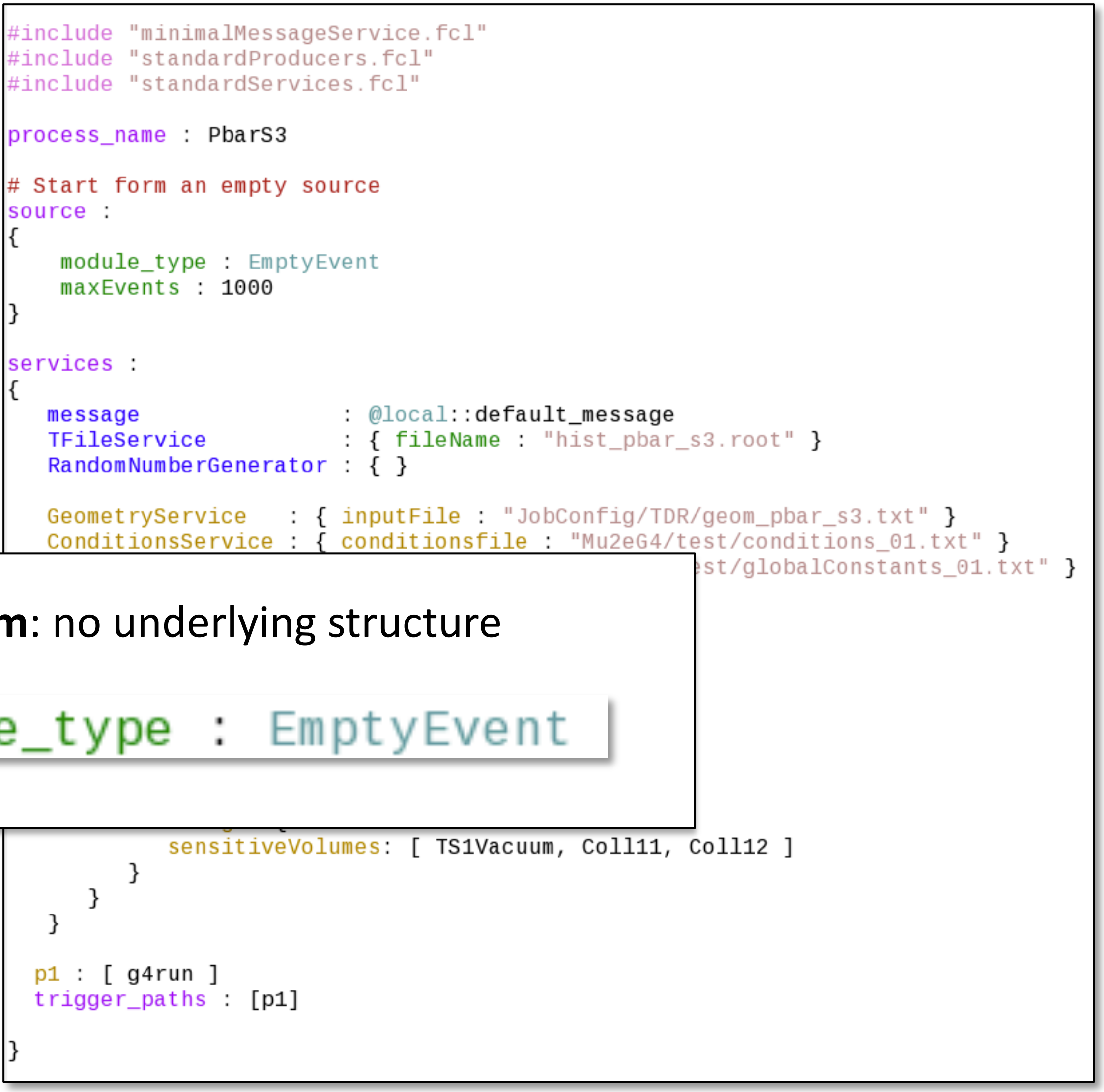
- *art* processes are configured using FHiCL files, that can look like this.



```
#include "minimalMessageService.fcl"
#include "standardProducers.fcl"
#include "standardServices.fcl"

process_name : PbarS3

# Start form an empty source
source :
{
    module_type : EmptyEvent
    maxEvents : 1000
}

services :
{
   message                : @local::default_message
   TFileService           : { fileName : "hist_pbar_s3.root" }
   RandomNumberGenerator : { }

   GeometryService   : { inputFile : "JobConfig/TDR/geom_pbar_s3.txt" }
   ConditionsService : { conditionsfile : "Mu2eG4/test/conditions_01.txt" }
   GlobalConstantsService : { inputFile : "Mu2eG4/test/globalConstants_01.txt" }
}

physics : {
   producers: {

      g4run : {
         module_type          : G4
         inputSimParticles    : "g4filter:s0"
         generatorModuleLabel : rotatetarget
         SDConfig : {
            sensitiveVolumes: [ TS1Vacuum, Coll11, Coll12 ]
         }
      }
   }

  p1 : [ g4run ]
  trigger_paths : [p1]

}
```

FHiCL and *art*

- *art* processes are configured using FHiCL files, that can look like this.



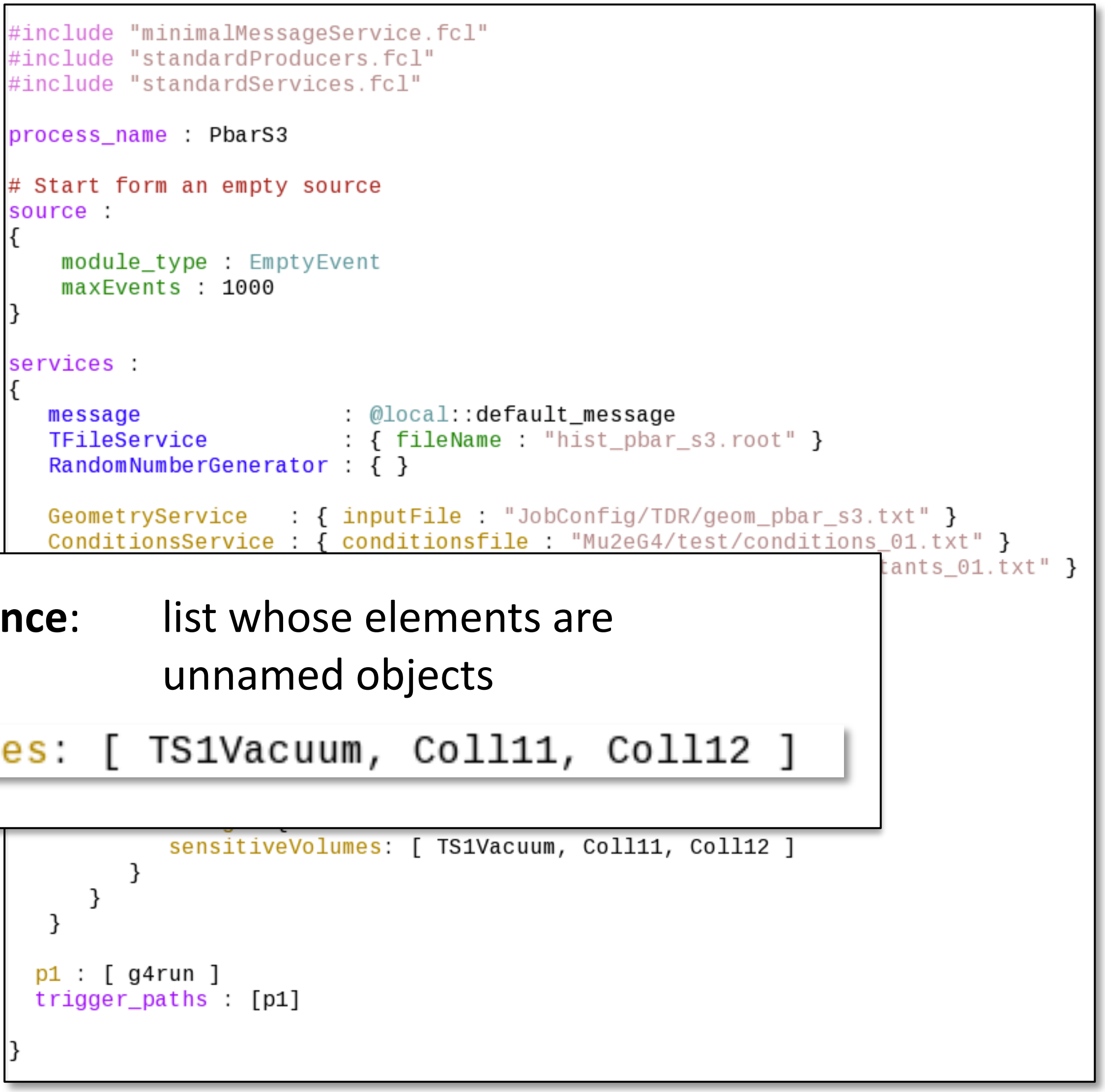
```
#include "minimalMessageService.fcl"
#include "standardProducers.fcl"
#include "standardServices.fcl"

process_name : PbarS3

# Start form an empty source
source :
{
    module_type : EmptyEvent
    maxEvents : 1000
}

services :
{
   message                 : @local::default_message
   TFileService            : { fileName : "hist_pbar_s3.root" }
   RandomNumberGenerator : { }

   GeometryService    : { inputFile : "JobConfig/TDR/geom_pbar_s3.txt" }
   ConditionsService : { conditionsfile : "Mu2eG4/test/conditions_01.txt" }
                                                             est/globalConstants_01.txt" }

            sensitiveVolumes: [ TS1Vacuum, Coll11, Coll12 ]
         }
      }
   }

  p1 : [ g4run ]
  trigger_paths : [p1]

}
```

Atom: no underlying structure

```
module_type : EmptyEvent
```

FHiCL and *art*

- *art* processes are configured using FHiCL files, that can look like this.



```
#include "minimalMessageService.fcl"
#include "standardProducers.fcl"
#include "standardServices.fcl"

process_name : PbarS3

# Start form an empty source
source :
{
    module_type : EmptyEvent
    maxEvents : 1000
}

services :
{
   message                 : @local::default_message
   TFileService            : { fileName : "hist_pbar_s3.root" }
   RandomNumberGenerator : { }

   GeometryService    : { inputFile : "JobConfig/TDR/geom_pbar_s3.txt" }
   ConditionsService : { conditionsfile : "Mu2eG4/test/conditions_01.txt" }
                                                               tants_01.txt" }
```

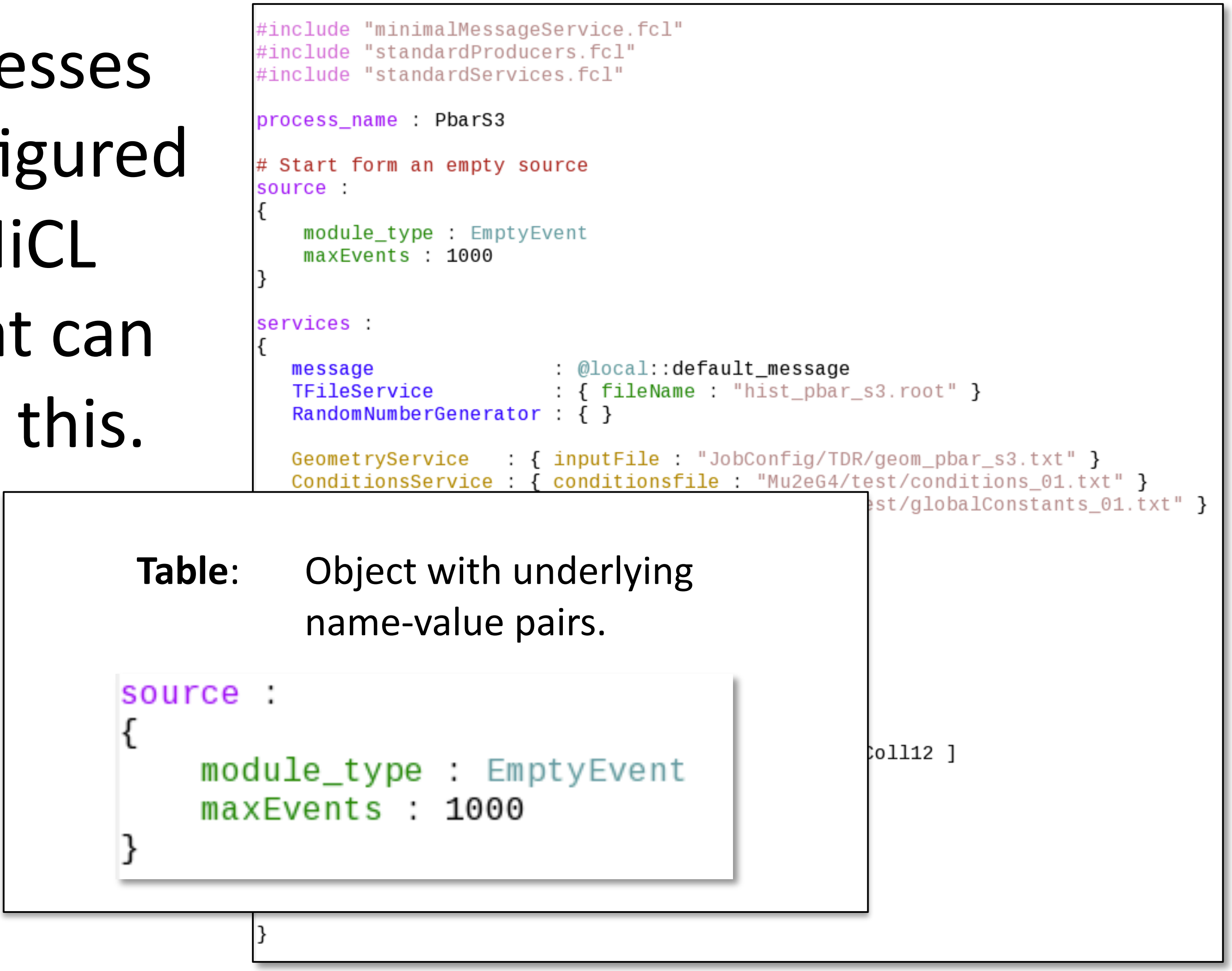
```
            sensitiveVolumes: [ TS1Vacuum, Coll11, Coll12 ]
         }
      }
   }

  p1 : [ g4run ]
  trigger_paths : [p1]

}
```

Sequence: list whose elements are unnamed objects

```
sensitiveVolumes: [ TS1Vacuum, Coll11, Coll12 ]
```

FHiCL and *art*

- *art* processes are configured using FHiCL files, that can look like this.



```
#include "minimalMessageService.fcl"
#include "standardProducers.fcl"
#include "standardServices.fcl"

process_name : PbarS3

# Start form an empty source
source :
{
    module_type : EmptyEvent
    maxEvents : 1000
}

services :
{
   message                 : @local::default_message
   TFileService            : { fileName : "hist_pbar_s3.root" }
   RandomNumberGenerator : { }

   GeometryService    : { inputFile : "JobConfig/TDR/geom_pbar_s3.txt" }
   ConditionsService : { conditionsfile : "Mu2eG4/test/conditions_01.txt" }
                                                   est/globalConstants_01.txt" }

                                                   Coll12 ]

}
```

Table: Object with underlying name-value pairs.

```
source :
{
    module_type : EmptyEvent
    maxEvents : 1000
}
```

FHiCL is very flexible

- The user decides how complicated the job configuration should be
  - Simpler configurations are better.
  - However, it can support very complicated nested structures.

- Retrieving parameter values in your source code is fairly straightforward (e.g.)

```
pset.get<int>(“someInt”)
pset.get<double>(“someTable.someDouble”);
```

- But the current system has limitations …

Some limitations

- Parameter misspellings are not noticed for parameters that have defaults.

- It is not possible to know which parameters are supported for a given module
  - i.e. no FHiCL description for a given module

- Parameter retrievals (i.e. pset.get<T>) can be awkward for structures more complicated than atoms.

ParameterSet validation goals

Validation

- Provide a means of validating configuration files against a specified reference for a given module.
  - Notifies user of parameters in *.fcl files that are not supported (e.g. misspellings fall into this category)
  - Notifies user of parameters that are ***missing*** from their FHiCL.

Description

- The specified reference must serve as a description so that users do not need to look at source code to determine the allowed configuration.

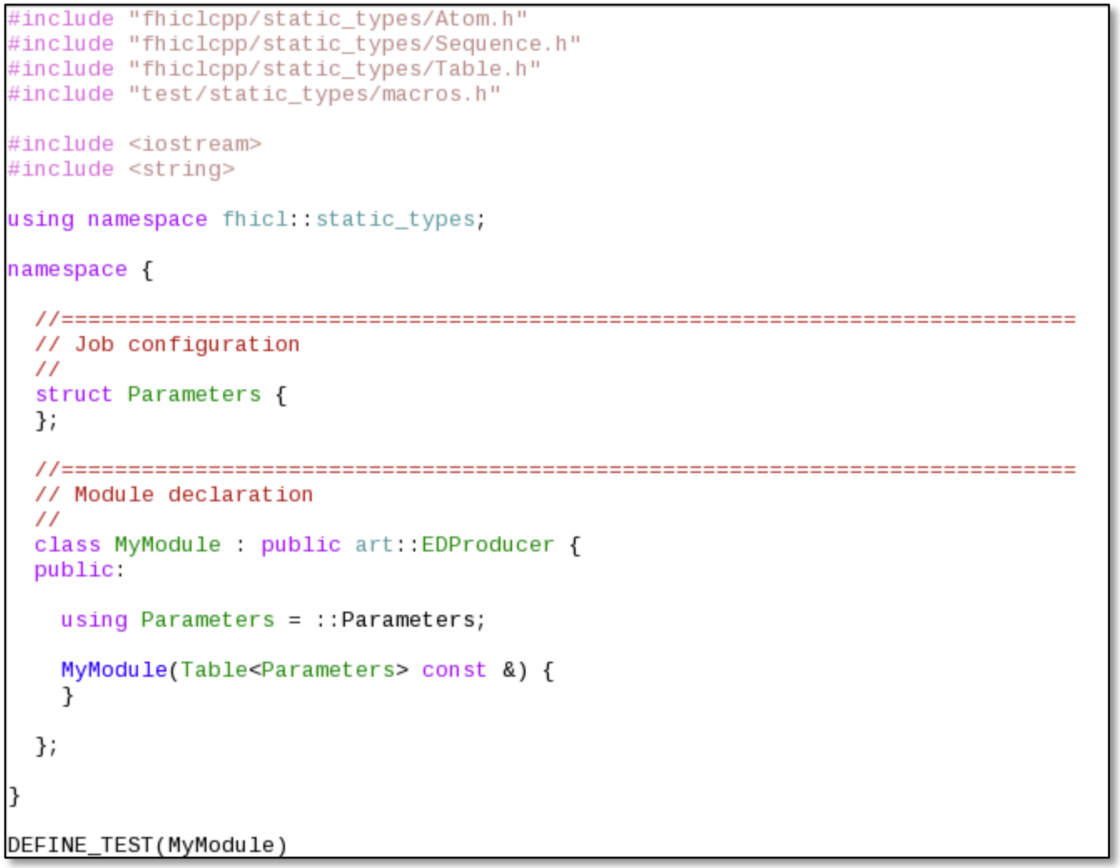
Ease of use

- User interface must be straightforward to understand and use.

Show and tell

- Today I’m using a toy module.

- This is very similar to what you would see when *art* constructing an *art* module.

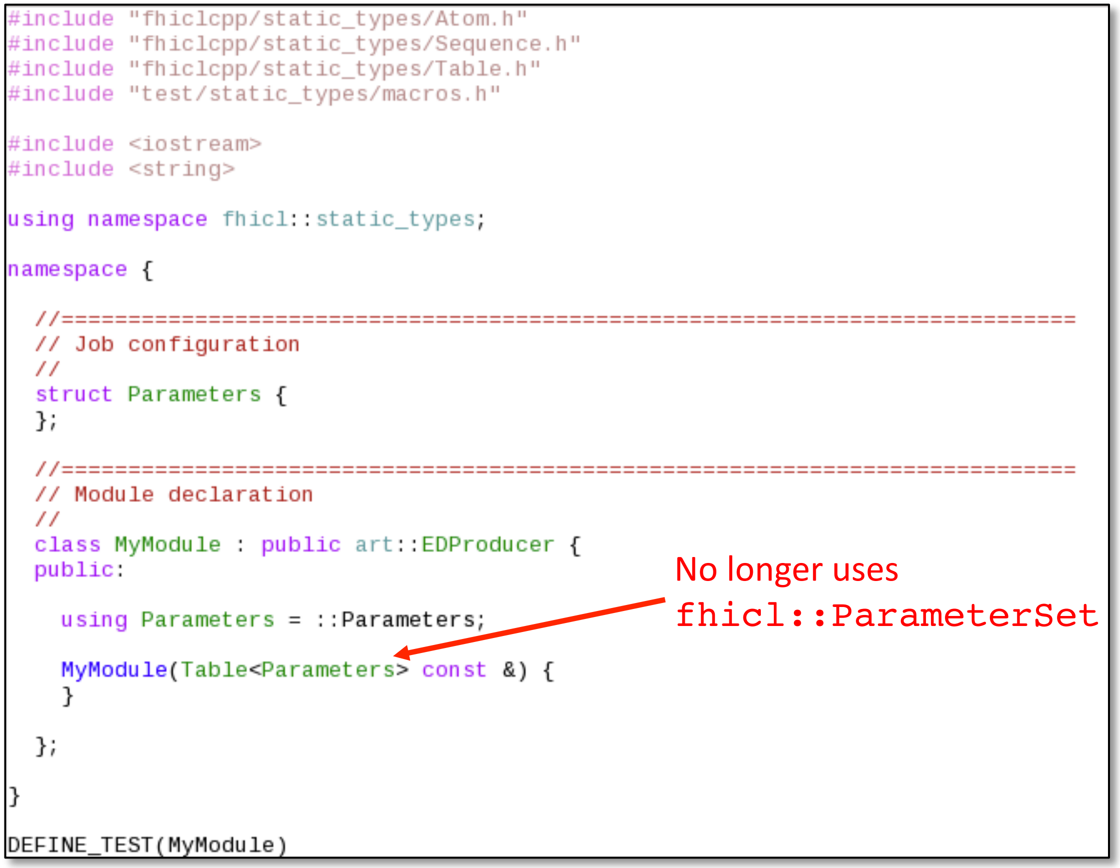


```
#include "fhiclcpp/static_types/Atom.h"
#include "fhiclcpp/static_types/Sequence.h"
#include "fhiclcpp/static_types/Table.h"
#include "test/static_types/macros.h"

#include <iostream>
#include <string>

using namespace fhicl::static_types;

namespace {

  //============================================================================
  // Job configuration
  //
  struct Parameters {
  };

  //============================================================================
  // Module declaration
  //
  class MyModule : public art::EDProducer {
  public:

    using Parameters = ::Parameters;

    MyModule(Table<Parameters> const &) {
    }

  };

}

DEFINE_TEST(MyModule)
```

Show and tell

- Today I’m using a toy module.

- This is very similar to what you would see when *art* constructing an *art* module.

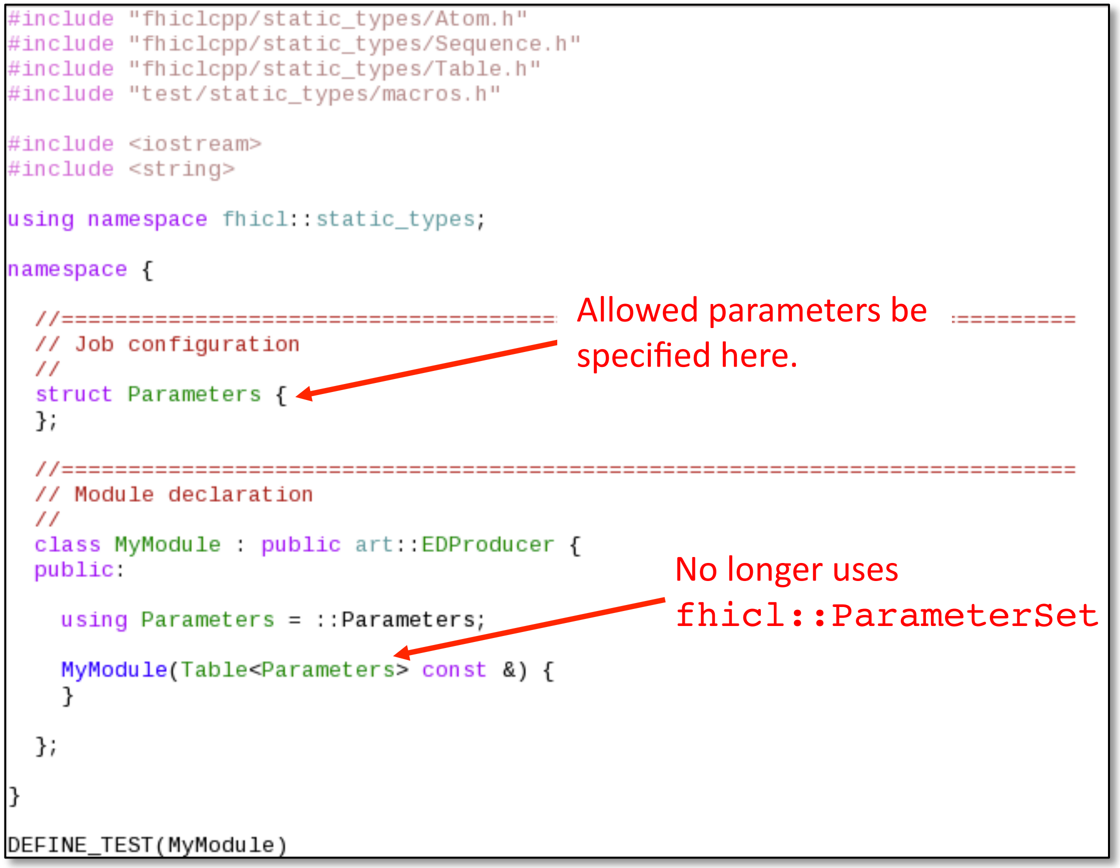


```
#include "fhiclcpp/static_types/Atom.h"
#include "fhiclcpp/static_types/Sequence.h"
#include "fhiclcpp/static_types/Table.h"
#include "test/static_types/macros.h"

#include <iostream>
#include <string>

using namespace fhicl::static_types;

namespace {

  //============================================================================
  // Job configuration
  //
  struct Parameters {
  };

  //============================================================================
  // Module declaration
  //
  class MyModule : public art::EDProducer {
  public:

    using Parameters = ::Parameters;

    MyModule(Table<Parameters> const &) {
    }

  };

}

DEFINE_TEST(MyModule)
```

No longer uses `fhicl::ParameterSet`

Show and tell

- Today I’m using a toy module.
- This is very similar to what you would see when *art* constructing an *art* module.

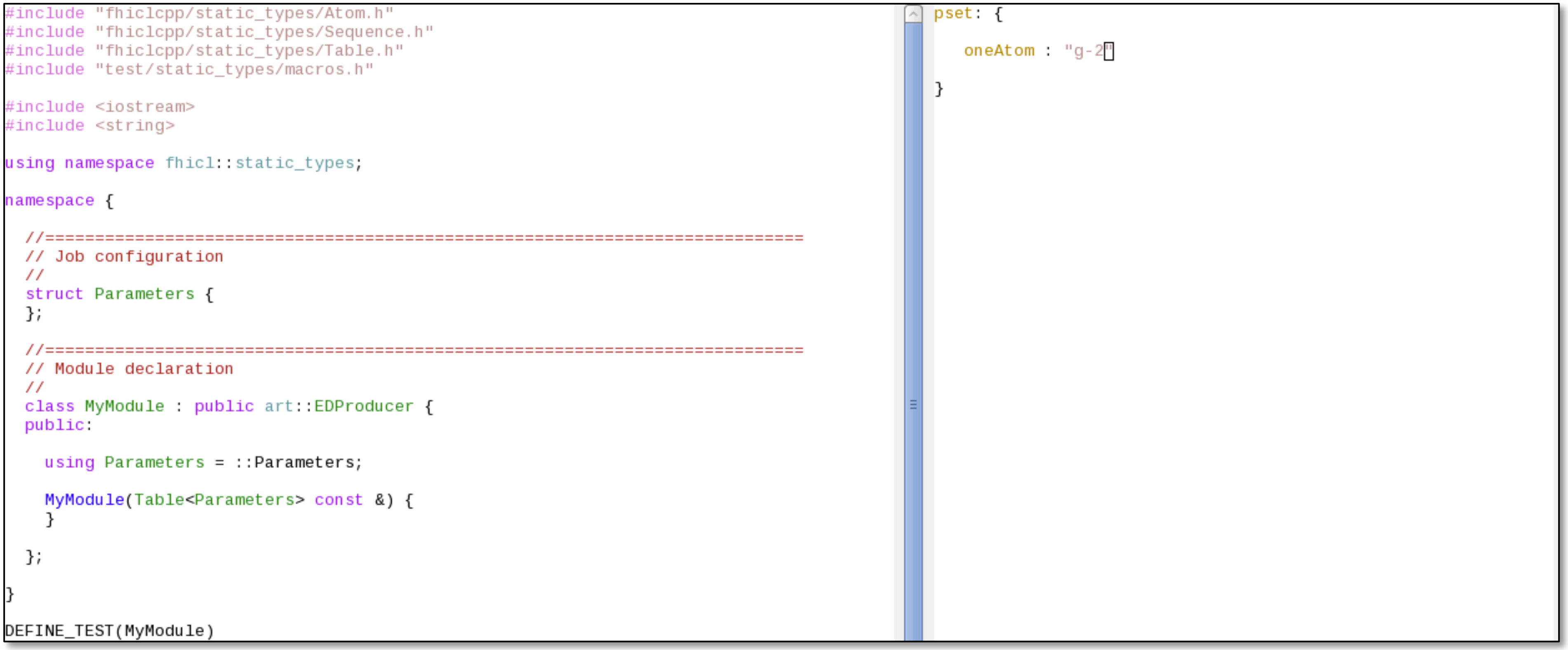


```
#include "fhiclcpp/static_types/Atom.h"
#include "fhiclcpp/static_types/Sequence.h"
#include "fhiclcpp/static_types/Table.h"
#include "test/static_types/macros.h"

#include <iostream>
#include <string>

using namespace fhicl::static_types;

namespace {

  //============================================================================
  // Job configuration
  //
  struct Parameters {
  };

  //============================================================================
  // Module declaration
  //
  class MyModule : public art::EDProducer {
  public:

    using Parameters = ::Parameters;

    MyModule(Table<Parameters> const &) {
    }

  };

}

DEFINE_TEST(MyModule)
```

Allowed parameters be specified here.

No longer uses `fhicl::ParameterSet`

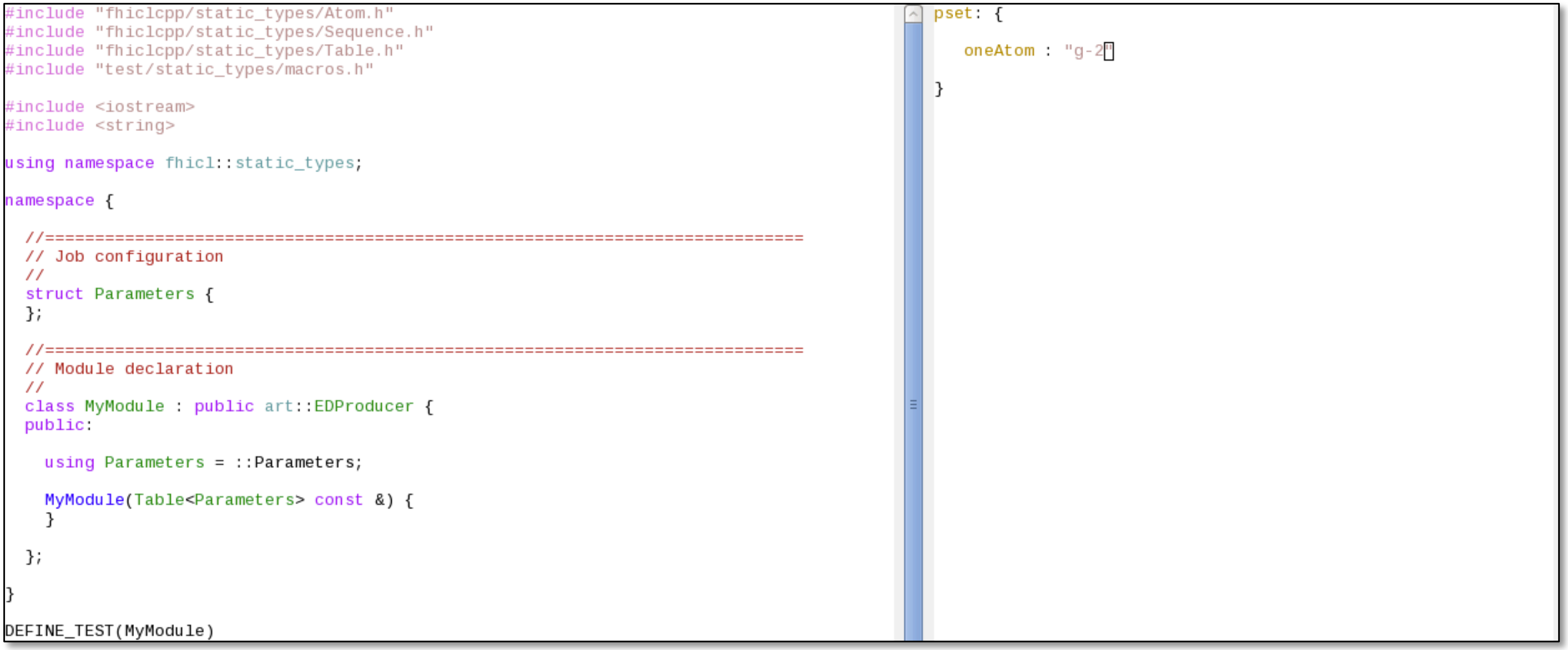
The simple case – one atom



```
#include "fhiclcpp/static_types/Atom.h"
#include "fhiclcpp/static_types/Sequence.h"
#include "fhiclcpp/static_types/Table.h"
#include "test/static_types/macros.h"

#include <iostream>
#include <string>

using namespace fhicl::static_types;

namespace {

  //============================================================================
  // Job configuration
  //
  struct Parameters {
  };

  //============================================================================
  // Module declaration
  //
  class MyModule : public art::EDProducer {
  public:

    using Parameters = ::Parameters;

    MyModule(Table<Parameters> const &) {
    }

  };

}

DEFINE_TEST(MyModule)
```

```
pset: {

   oneAtom : "g-2"

}
```

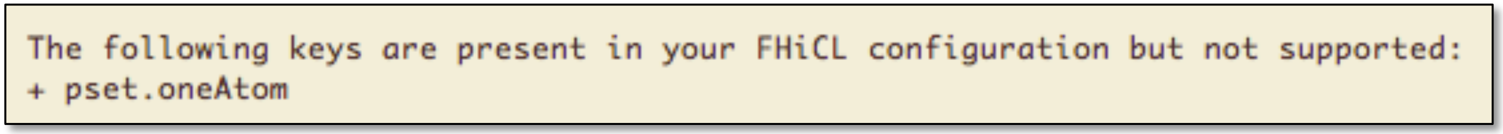
The simple case – one atom



```cpp
#include "fhiclcpp/static_types/Atom.h"
#include "fhiclcpp/static_types/Sequence.h"
#include "fhiclcpp/static_types/Table.h"
#include "test/static_types/macros.h"

#include <iostream>
#include <string>

using namespace fhicl::static_types;

namespace {

  //============================================================================
  // Job configuration
  //
  struct Parameters {
  };

  //============================================================================
  // Module declaration
  //
  class MyModule : public art::EDProducer {
  public:

    using Parameters = ::Parameters;

    MyModule(Table<Parameters> const &) {
    }

  };

}

DEFINE_TEST(MyModule)
```

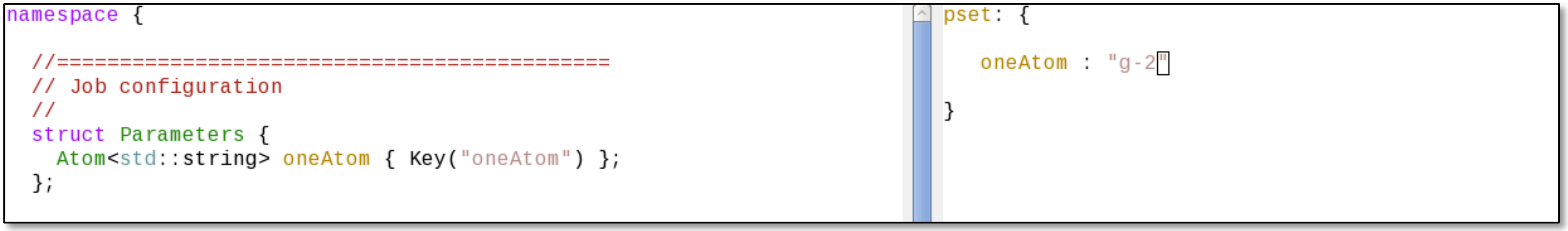
```
pset: {

   oneAtom : "g-2"

}
```

Loading module:



```
The following keys are present in your FHiCL configuration but not supported:
+ pset.oneAtom
```

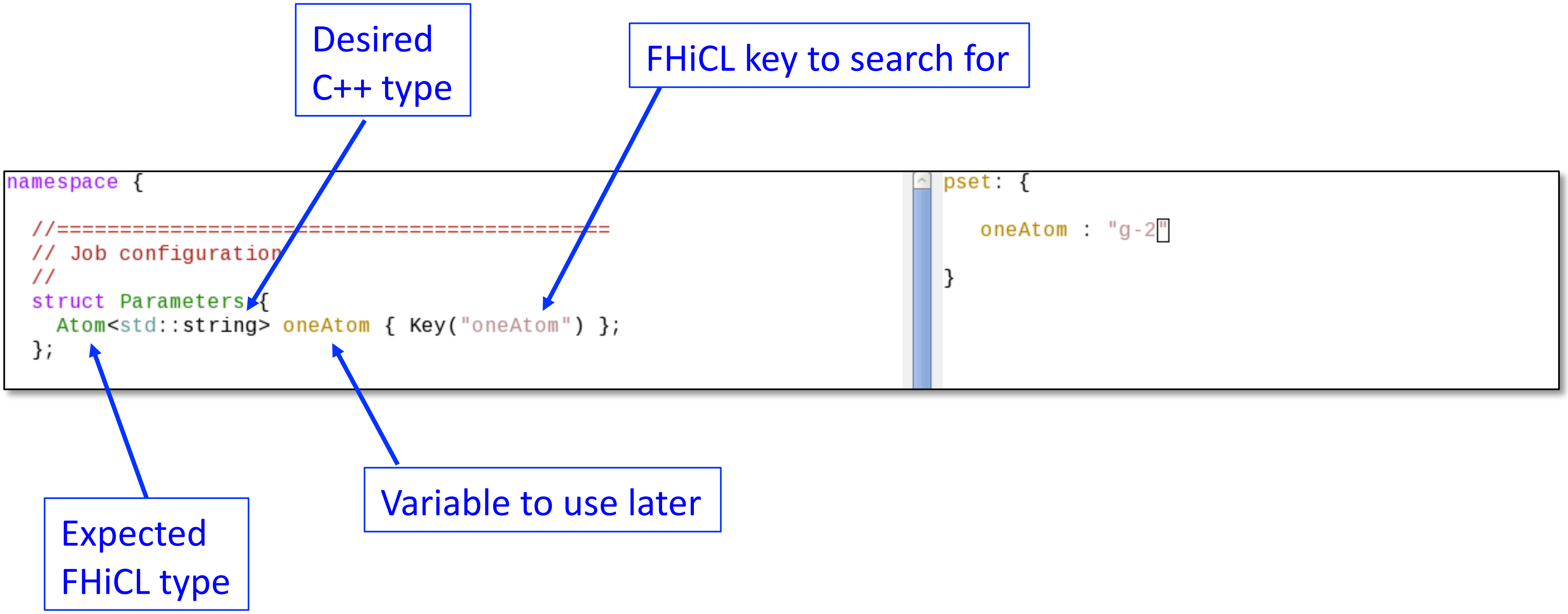
The simple case – one atom



```
namespace {

  //============================================
  // Job configuration
  //
  struct Parameters {
    Atom<std::string> oneAtom { Key("oneAtom") };
  };
```

```
pset: {

   oneAtom : "g-2"

}
```

The simple case – one atom



```
namespace {

  //=============================================
  // Job configuration
  //
  struct Parameters {
    Atom<std::string> oneAtom { Key("oneAtom") };
  };
```

```
pset: {

   oneAtom : "g-2"

}
```

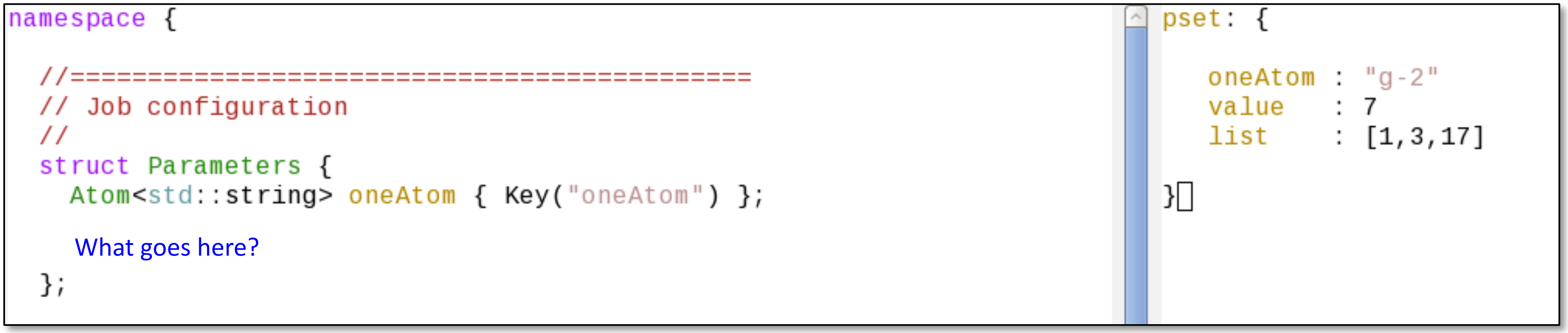
Desired C++ type

FHiCL key to search for

Expected FHiCL type

Variable to use later

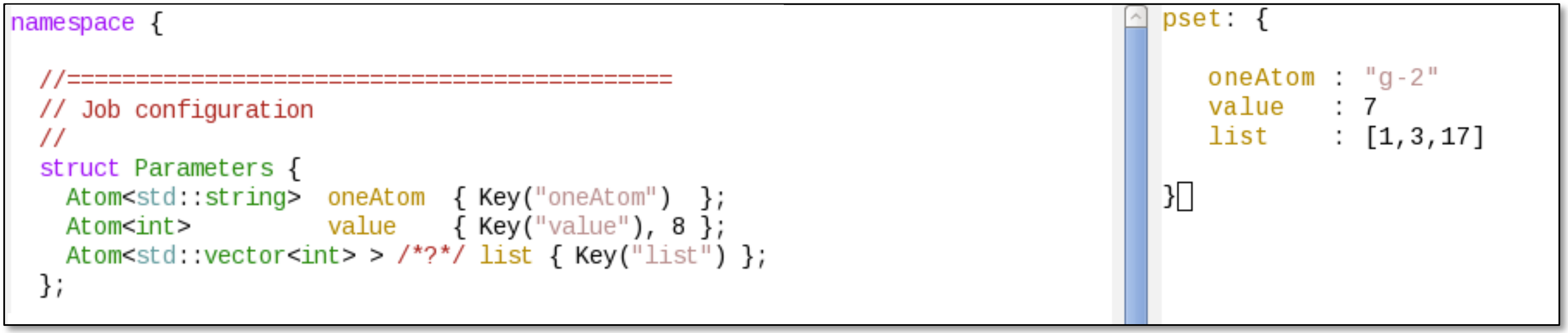
Adding some structure



```
namespace {

  //============================================
  // Job configuration
  //
  struct Parameters {
    Atom<std::string> oneAtom { Key("oneAtom") };

     What goes here?
  };
```

```
pset: {

   oneAtom : "g-2"
   value   : 7
   list    : [1,3,17]

}
```

Adding some structure

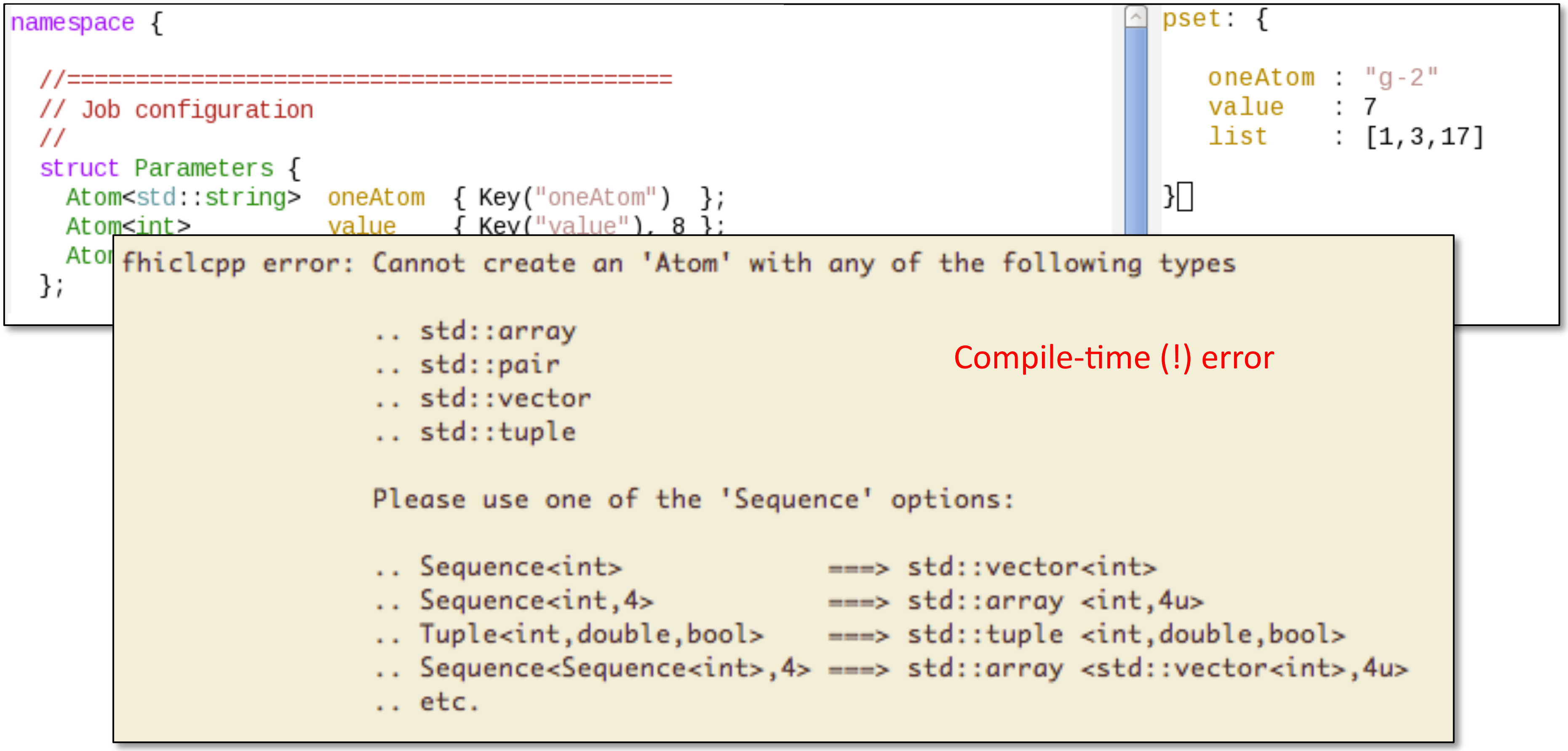


```
namespace {

  //============================================
  // Job configuration
  //
  struct Parameters {
    Atom<std::string>  oneAtom  { Key("oneAtom")  };
    Atom<int>          value    { Key("value"), 8 };
    Atom<std::vector<int> > /*?*/ list { Key("list") };
  };
```

```
pset: {

    oneAtom  : "g-2"
    value    : 7
    list     : [1,3,17]

}
```

If you compile you get …

Adding some structure



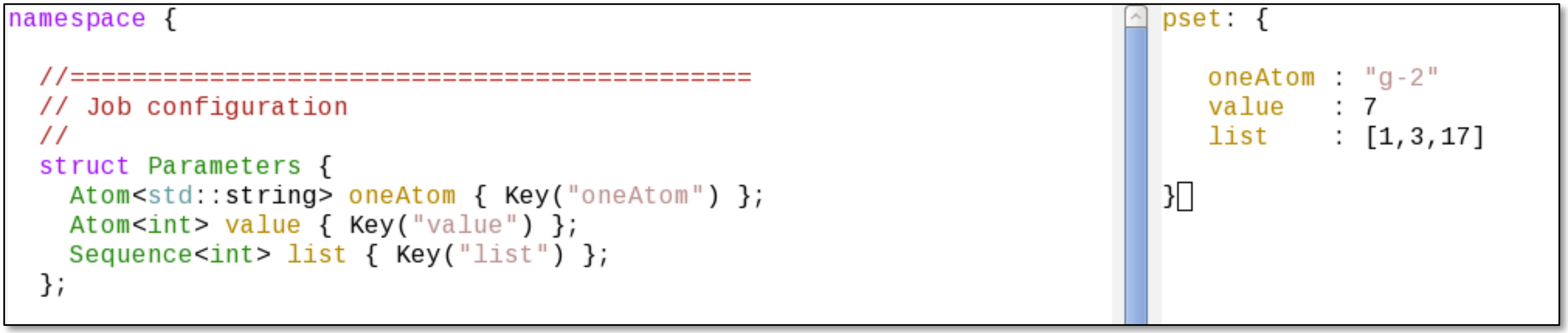
```
namespace {

  //============================================
  // Job configuration
  //
  struct Parameters {
    Atom<std::string>  oneAtom  { Key("oneAtom")  };
    Atom<int>          value    { Key("value"), 8 };
    Ato
  };
```

```
pset: {

   oneAtom : "g-2"
   value   : 7
   list    : [1,3,17]

}
```

```
fhiclcpp error: Cannot create an 'Atom' with any of the following types

                .. std::array
                .. std::pair
                .. std::vector
                .. std::tuple

                Please use one of the 'Sequence' options:

                .. Sequence<int>              ===> std::vector<int>
                .. Sequence<int,4>            ===> std::array <int,4u>
                .. Tuple<int,double,bool>     ===> std::tuple <int,double,bool>
                .. Sequence<Sequence<int>,4>  ===> std::array <std::vector<int>,4u>
                .. etc.
```

Compile-time (!) error

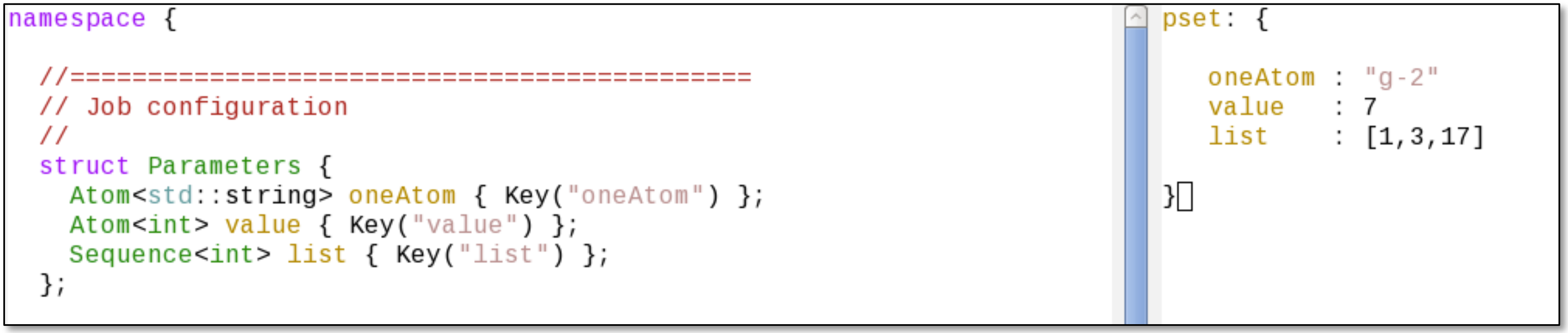
Including sequences



```
namespace {

  //==============================================
  // Job configuration
  //
  struct Parameters {
    Atom<std::string> oneAtom { Key("oneAtom") };
    Atom<int> value { Key("value") };
    Sequence<int> list { Key("list") };
  };
```

```
pset: {

   oneAtom : "g-2"
   value   : 7
   list    : [1,3,17]

}
```

Including sequences



```
namespace {

  //=============================================
  // Job configuration
  //
  struct Parameters {
    Atom<std::string> oneAtom { Key("oneAtom") };
    Atom<int> value { Key("value") };
    Sequence<int> list { Key("list") };
  };
```

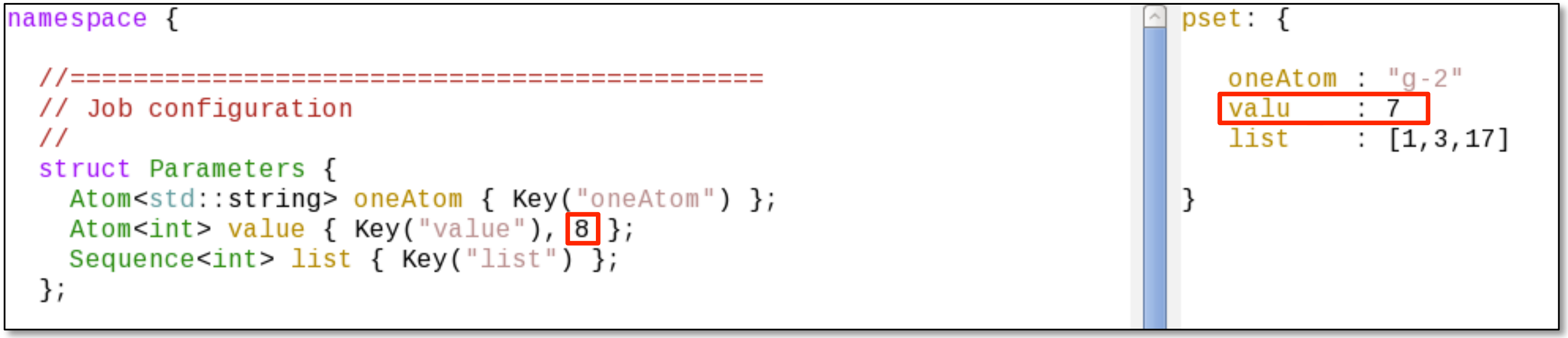
```
pset: {

   oneAtom : "g-2"
   value   : 7
   list    : [1,3,17]

}
```

- Three kinds of containers that support sequences (e.g.):

fhiclcpp type	**Underlying std:: type**
Sequence<int>	std::vector<int>
Sequence<int,4>	std::array<int,4>
Tuple<int,double>	std::tuple<int,double>

Introducing defaults

Attempt to
override default

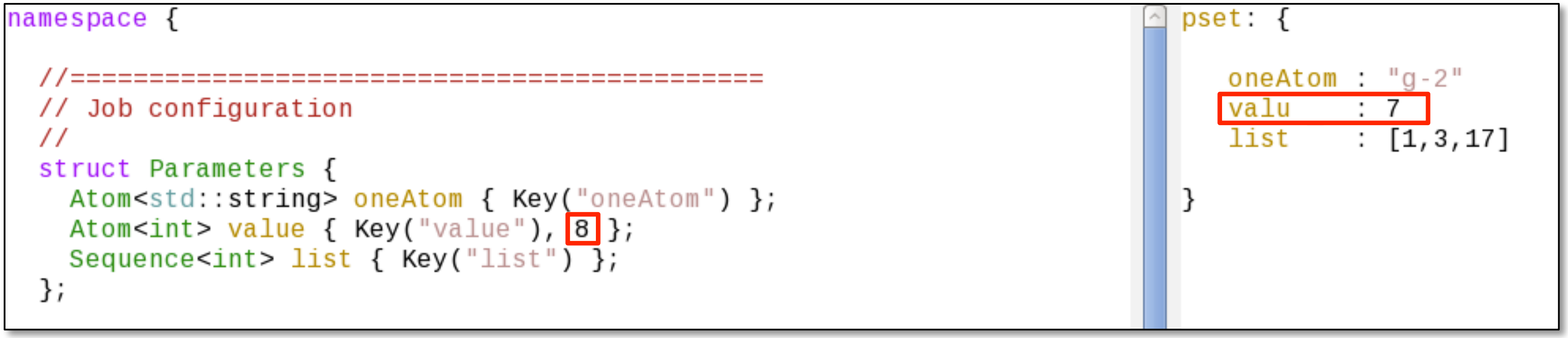


```
namespace {

  //==============================================
  // Job configuration
  //
  struct Parameters {
    Atom<std::string> oneAtom { Key("oneAtom") };
    Atom<int> value { Key("value"), 8 };
    Sequence<int> list { Key("list") };
  };
```

```
pset: {

   oneAtom  : "g-2"
   valu     : 7
   list     : [1,3,17]

}
```

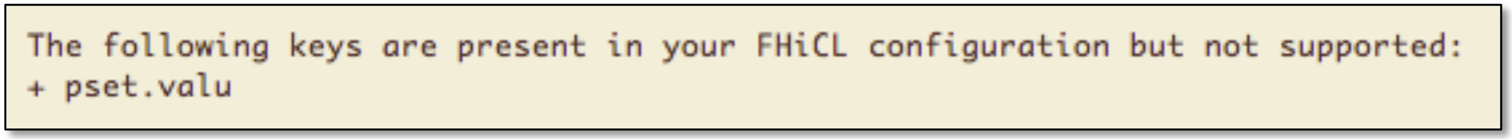
Introducing defaults

Attempt to override default



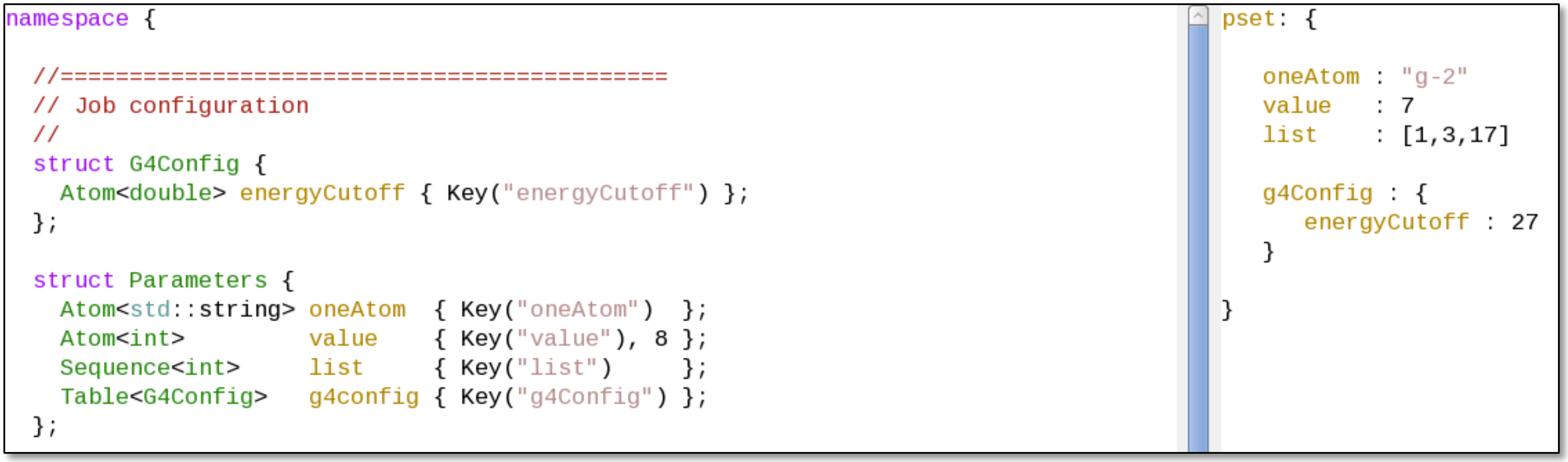
```
namespace {

  //=============================================
  // Job configuration
  //
  struct Parameters {
    Atom<std::string> oneAtom { Key("oneAtom") };
    Atom<int> value { Key("value"), 8 };
    Sequence<int> list { Key("list") };
  };
```

```
pset: {

   oneAtom  :  "g-2"
   valu     :  7
   list     :  [1,3,17]

}
```



```
The following keys are present in your FHiCL configuration but not supported:
+ pset.valu
```

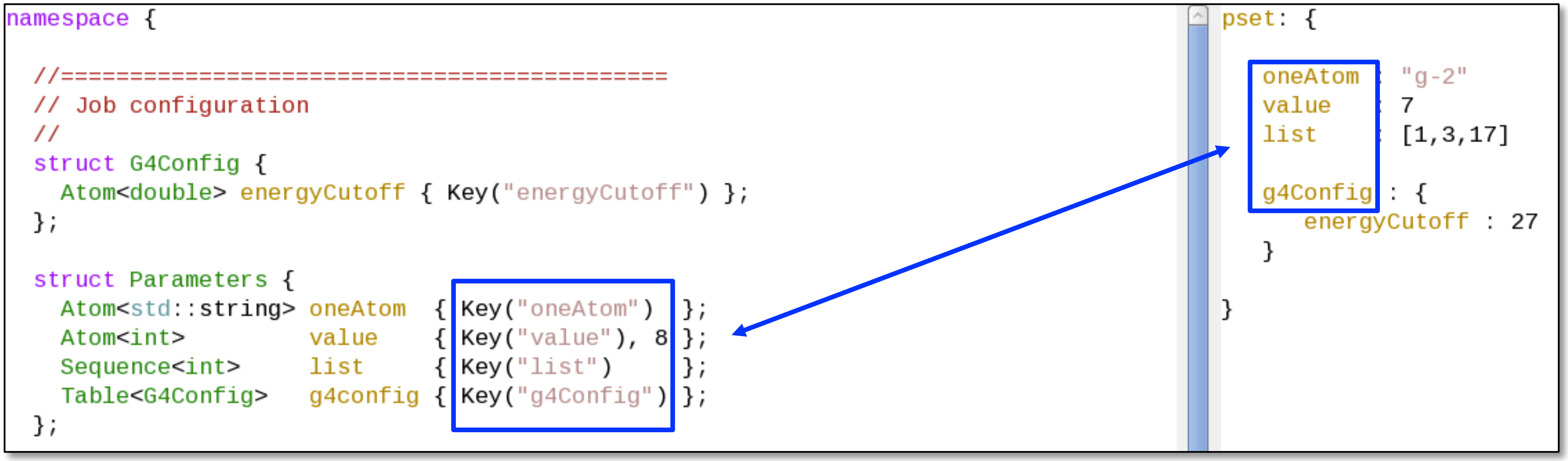
Introducing nested tables



```
namespace {

  //============================================
  // Job configuration
  //
  struct G4Config {
    Atom<double> energyCutoff { Key("energyCutoff") };
  };

  struct Parameters {
    Atom<std::string> oneAtom  { Key("oneAtom")  };
    Atom<int>         value    { Key("value"), 8 };
    Sequence<int>     list     { Key("list")     };
    Table<G4Config>   g4config { Key("g4Config") };
  };
```

```
pset: {

   oneAtom : "g-2"
   value   : 7
   list    : [1,3,17]

   g4Config : {
      energyCutoff : 27
   }

}
```

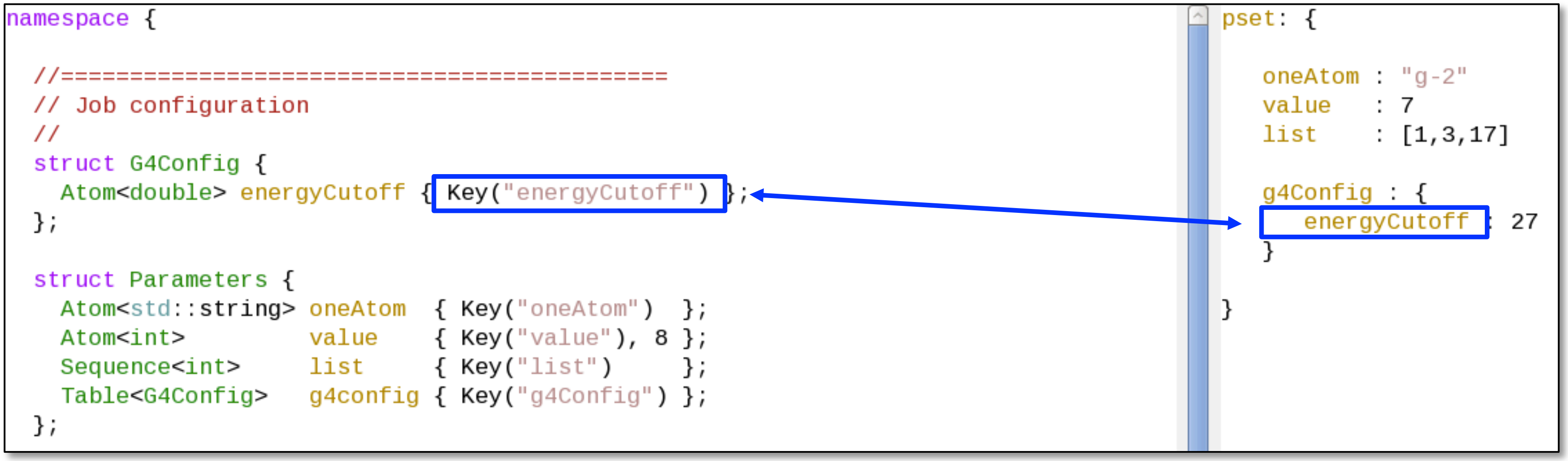
Introducing nested tables



```
namespace {

  //==============================================
  // Job configuration
  //
  struct G4Config {
    Atom<double> energyCutoff { Key("energyCutoff") };
  };

  struct Parameters {
    Atom<std::string> oneAtom   { Key("oneAtom")    };
    Atom<int>         value     { Key("value"), 8 };
    Sequence<int>     list      { Key("list")       };
    Table<G4Config>   g4config { Key("g4Config") };
  };
```

```
pset: {

   oneAtom  : "g-2"
   value    : 7
   list     : [1,3,17]

   g4Config : {
      energyCutoff : 27
   }

}
```

Introducing nested tables

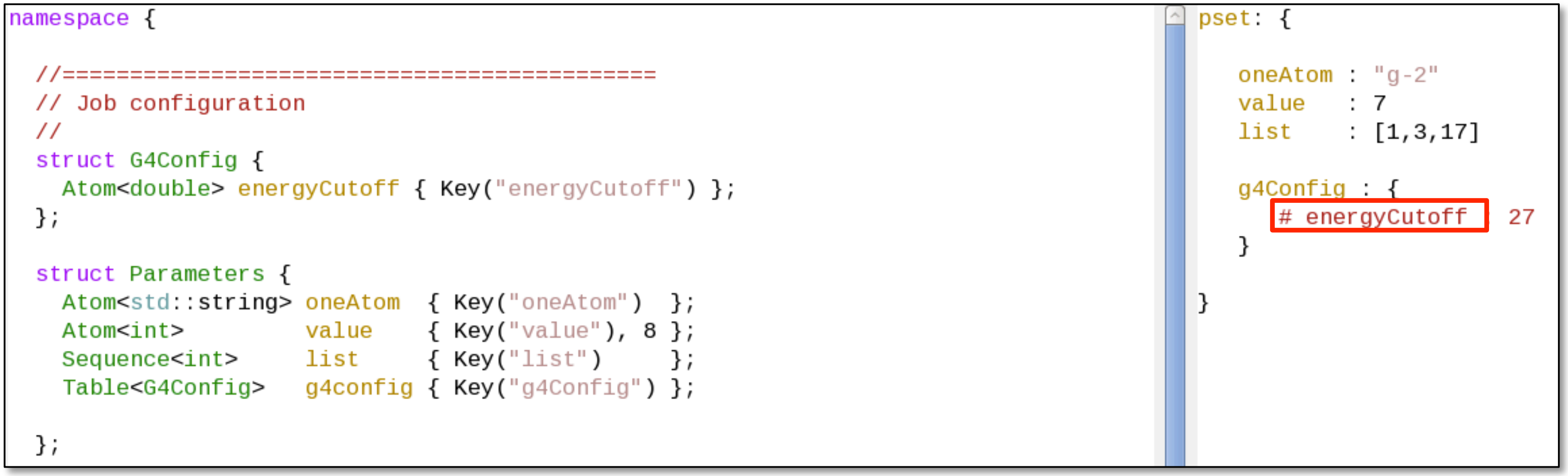


```
namespace {

  //=============================================
  // Job configuration
  //
  struct G4Config {
    Atom<double> energyCutoff { Key("energyCutoff") };
  };

  struct Parameters {
    Atom<std::string> oneAtom  { Key("oneAtom")  };
    Atom<int>         value    { Key("value"), 8 };
    Sequence<int>     list     { Key("list")     };
    Table<G4Config>   g4config { Key("g4Config") };
  };
```

```
pset: {

   oneAtom : "g-2"
   value   : 7
   list    : [1,3,17]

   g4Config : {
      energyCutoff : 27
   }

}
```

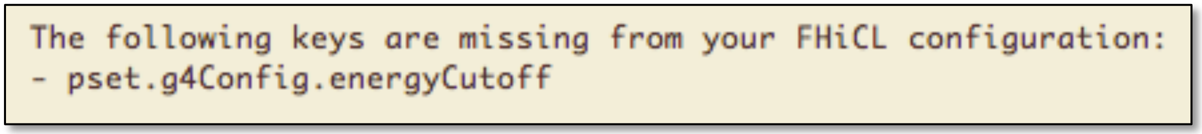
Introducing nested tables

Omitting key
that's expected



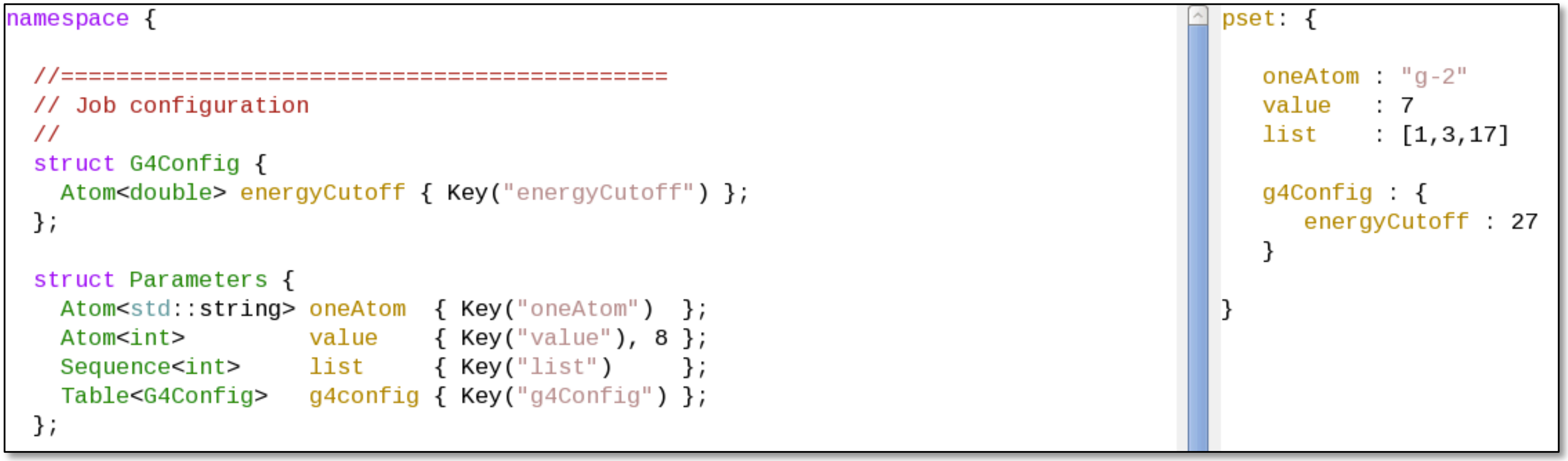
```
namespace {

  //==========================================
  // Job configuration
  //
  struct G4Config {
    Atom<double> energyCutoff { Key("energyCutoff") };
  };

  struct Parameters {
    Atom<std::string> oneAtom  { Key("oneAtom")  };
    Atom<int>         value    { Key("value"), 8 };
    Sequence<int>     list     { Key("list")     };
    Table<G4Config>   g4config { Key("g4Config") };

  };
```

```
pset: {

   oneAtom  : "g-2"
   value    : 7
   list     : [1,3,17]

   g4Config : {
     # energyCutoff    27
   }

}
```



```
The following keys are missing from your FHiCL configuration:
- pset.g4Config.energyCutoff
```

Introducing nested tables



```
namespace {

  //============================================
  // Job configuration
  //
  struct G4Config {
    Atom<double> energyCutoff { Key("energyCutoff") };
  };

  struct Parameters {
    Atom<std::string> oneAtom  { Key("oneAtom")  };
    Atom<int>         value    { Key("value"), 8 };
    Sequence<int>     list     { Key("list")     };
    Table<G4Config>   g4config { Key("g4Config") };
  };
```

```
pset: {

   oneAtom : "g-2"
   value   : 7
   list    : [1,3,17]

   g4Config : {
      energyCutoff : 27
   }

}
```

- From these tools, the C++ source can describe any FHiCL parameter set.

Allowed types (e.g.)

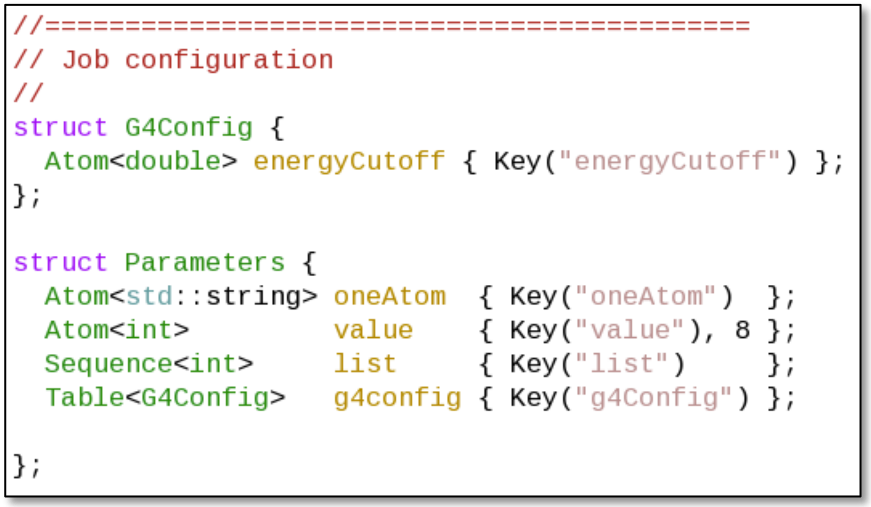
```
Atom<T>

Sequence<T>

Sequence<T,SZ>

Tuple<T…>

Tuple< Sequence<T>, U… >

Tuple< Sequence<T,SZ>, U… >

Tuple< Tuple<T…>,U…>

Sequence< Tuple<T…> >

Sequence< Tuple<T…>, SZ >

Sequence< Sequence<T> >

Sequence< Sequence<T,SZ> >
```

```
Sequence< Sequence<T>, SZ >

Sequence< Sequence<T,SZ>, SZ >

Table<S>

Sequence< Table<S> >

Sequence< Table<S>, SZ >

Tuple< Table<S>, U… >

Tuple< Sequence< Table<S> >, U… >

Tuple< Sequence< Table<S>, SZ>, U… >

Sequence< Tuple< Table<S>, U… > >

Sequence< Tuple< Table<S>, U… >, SZ>
```

N.B. List is meant to illustrate flexibility of system,
not to encourage the use of complicated types.

Print description of expected parameters

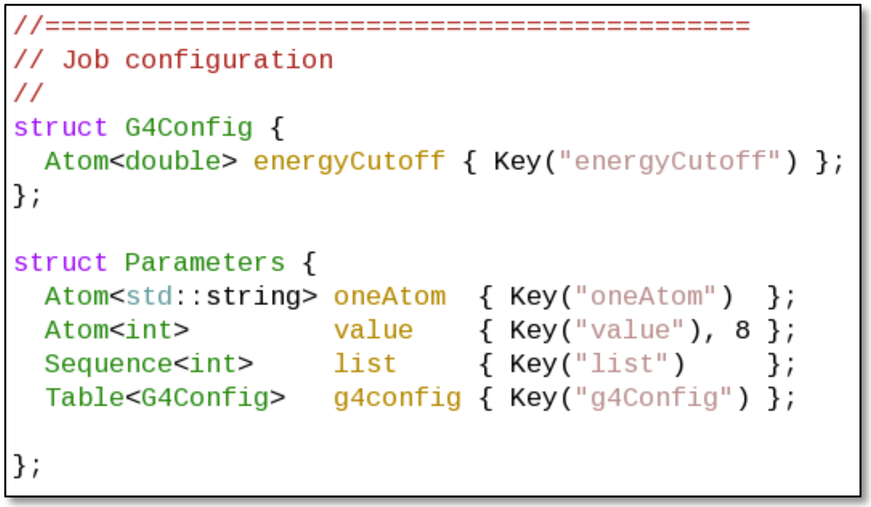
`art --module-description MyModule`



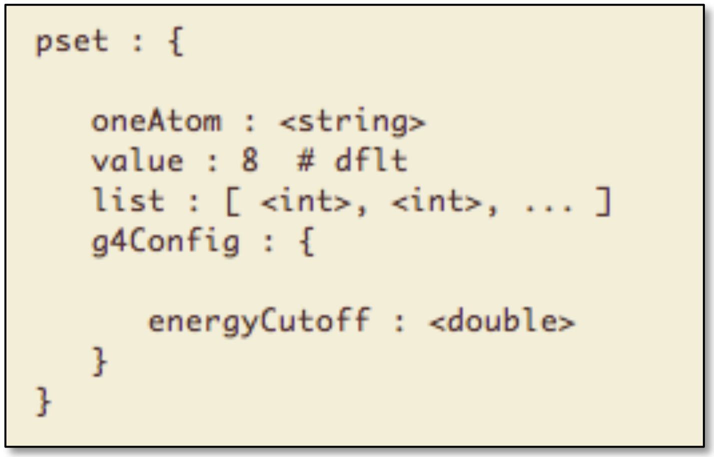
```
//============================================
// Job configuration
//
struct G4Config {
  Atom<double> energyCutoff { Key("energyCutoff") };
};

struct Parameters {
  Atom<std::string> oneAtom  { Key("oneAtom")  };
  Atom<int>         value    { Key("value"), 8 };
  Sequence<int>     list     { Key("list")     };
  Table<G4Config>   g4config { Key("g4Config") };

};
```

Print description of expected parameters

```
art --module-description MyModule
```



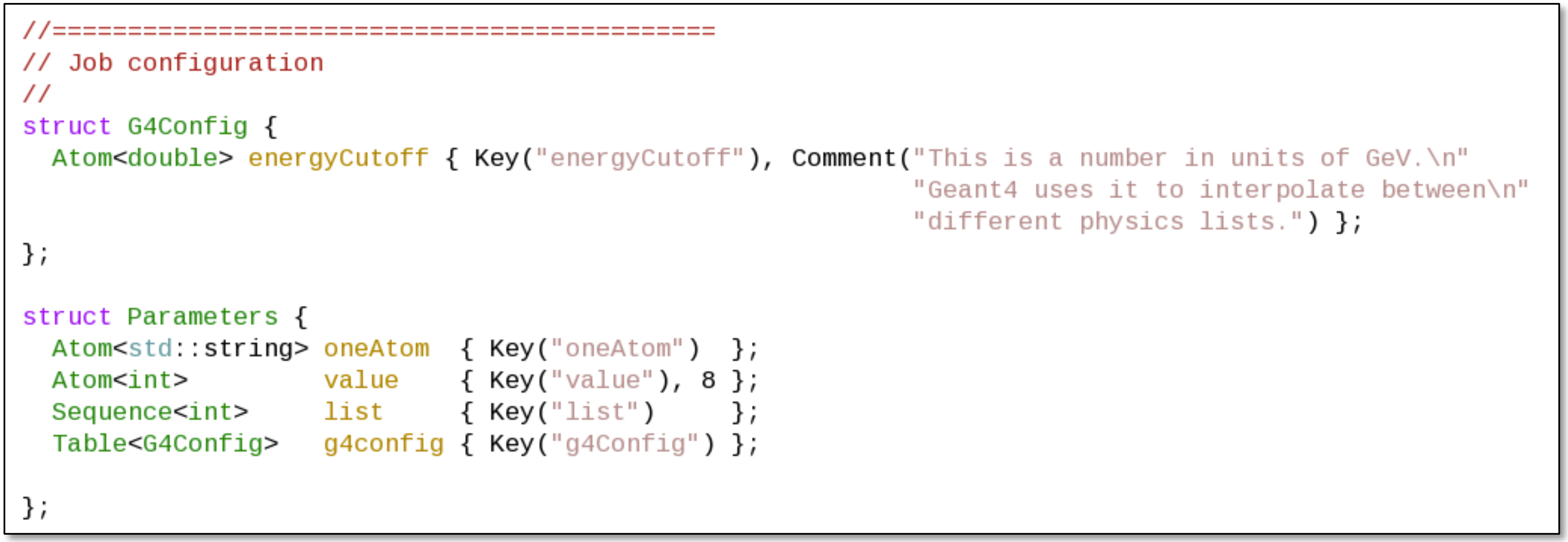
```
//=============================================
// Job configuration
//
struct G4Config {
  Atom<double> energyCutoff { Key("energyCutoff") };
};

struct Parameters {
  Atom<std::string> oneAtom  { Key("oneAtom")  };
  Atom<int>         value    { Key("value"), 8 };
  Sequence<int>     list     { Key("list")     };
  Table<G4Config>   g4config { Key("g4Config") };

};
```



```
pset : {

   oneAtom : <string>
   value : 8  # dflt
   list : [ <int>, <int>, ... ]
   g4Config : {

      energyCutoff : <double>
   }
}
```

- This is a significant improvement from where things stand now; but I still don’t know what the (e.g.) “energyCutoff” is.

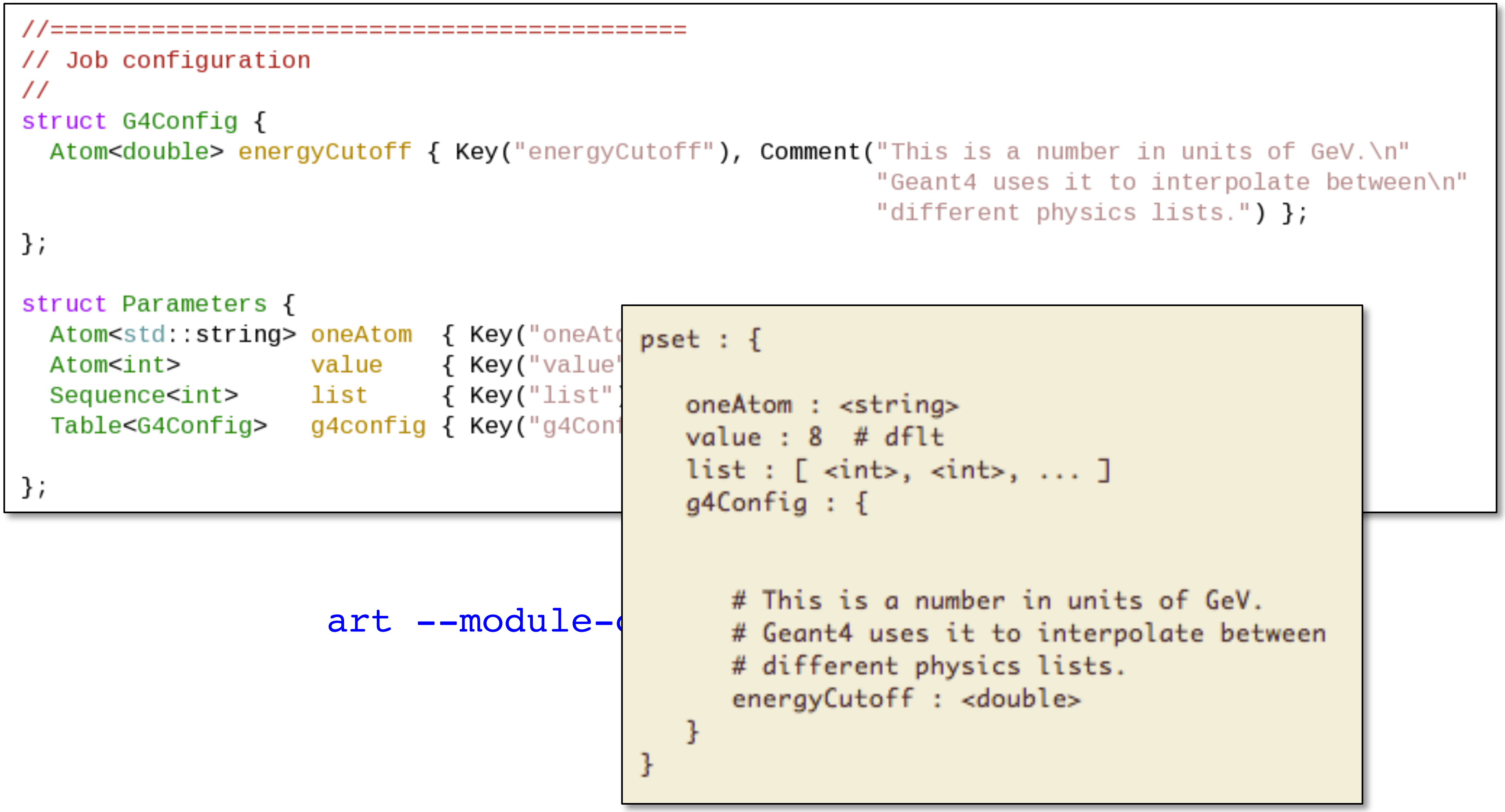
Introducing Comment



```
//============================================
// Job configuration
//
struct G4Config {
  Atom<double> energyCutoff { Key("energyCutoff"), Comment("This is a number in units of GeV.\n"
                                                           "Geant4 uses it to interpolate between\n"
                                                           "different physics lists.") };

};

struct Parameters {
  Atom<std::string> oneAtom  { Key("oneAtom")  };
  Atom<int>         value    { Key("value"), 8 };
  Sequence<int>     list     { Key("list")     };
  Table<G4Config>   g4config { Key("g4Config") };

};
```

`art --module-description MyModule` yields …

Introducing Comment



```
//============================================
// Job configuration
//
struct G4Config {
  Atom<double> energyCutoff { Key("energyCutoff"), Comment("This is a number in units of GeV.\n"
                                                             "Geant4 uses it to interpolate between\n"
                                                             "different physics lists.") };

};

struct Parameters {
  Atom<std::string> oneAtom  { Key("oneAt
  Atom<int>         value    { Key("value'
  Sequence<int>     list     { Key("list")
  Table<G4Config>   g4config { Key("g4Conf

};
```

```
pset : {

   oneAtom : <string>
   value : 8  # dflt
   list : [ <int>, <int>, ... ]
   g4Config : {


      # This is a number in units of GeV.
      # Geant4 uses it to interpolate between
      # different physics lists.
      energyCutoff : <double>
   }
}
```

```
art --module-
```

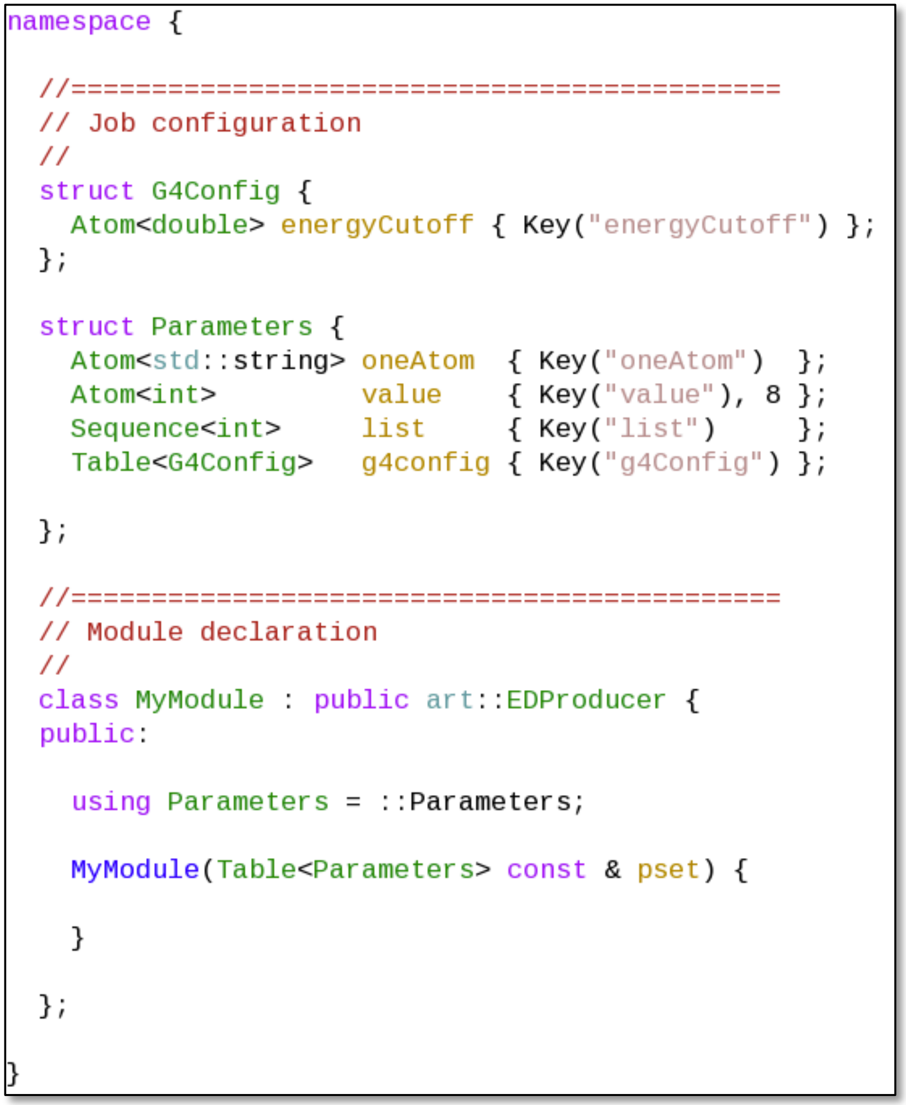
Accessing the elements

- After the *.fcl file has been validated, the module is constructed and users can access the elements.

Accessing the elements

- After the *.fcl file has been validated, the module is constructed and users can access the elements.

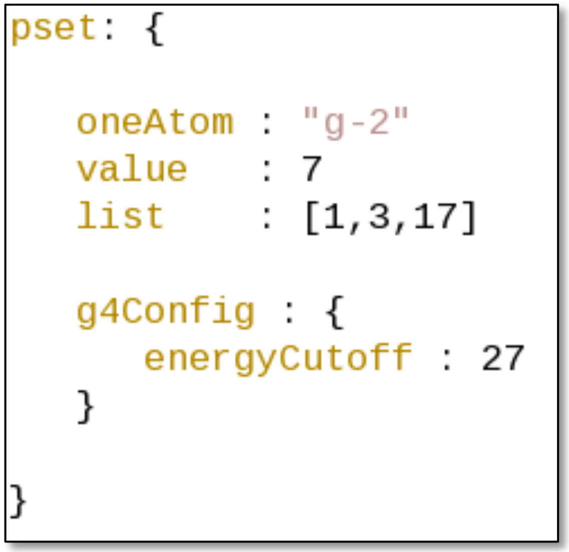
- Reminder of our module:



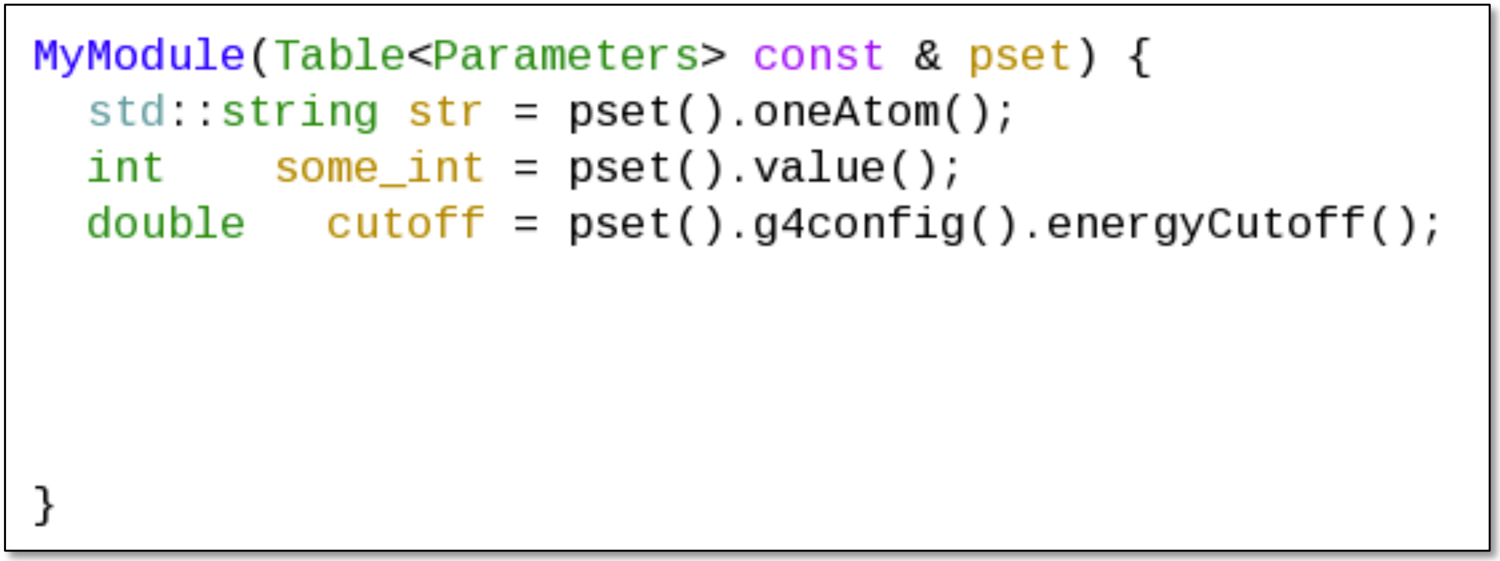
```
namespace {

  //==========================================
  // Job configuration
  //
  struct G4Config {
    Atom<double> energyCutoff { Key("energyCutoff") };
  };

  struct Parameters {
    Atom<std::string> oneAtom  { Key("oneAtom")  };
    Atom<int>         value    { Key("value"), 8 };
    Sequence<int>     list     { Key("list")     };
    Table<G4Config>   g4config { Key("g4Config") };

  };

  //==========================================
  // Module declaration
  //
  class MyModule : public art::EDProducer {
  public:

    using Parameters = ::Parameters;

    MyModule(Table<Parameters> const & pset) {

    }

  };

}
```

Accessing the elements

- No more `pset.get<>`.
  - Accessing elements is now done using syntax very similar to FHiCL syntax.



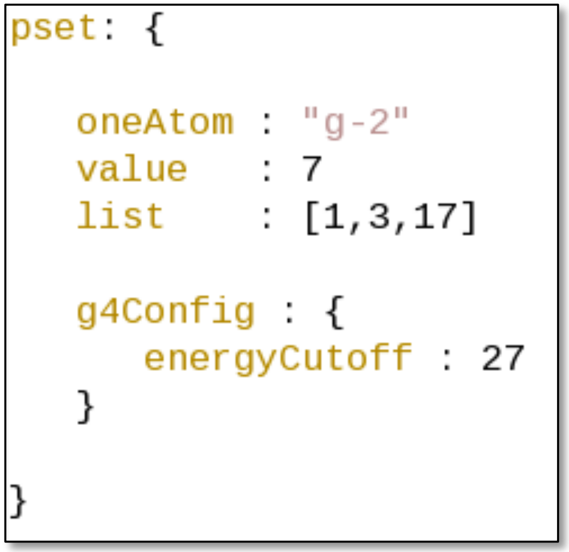
```
pset: {

   oneAtom : "g-2"
   value   : 7
   list    : [1,3,17]

   g4Config : {
      energyCutoff : 27
   }

}
```



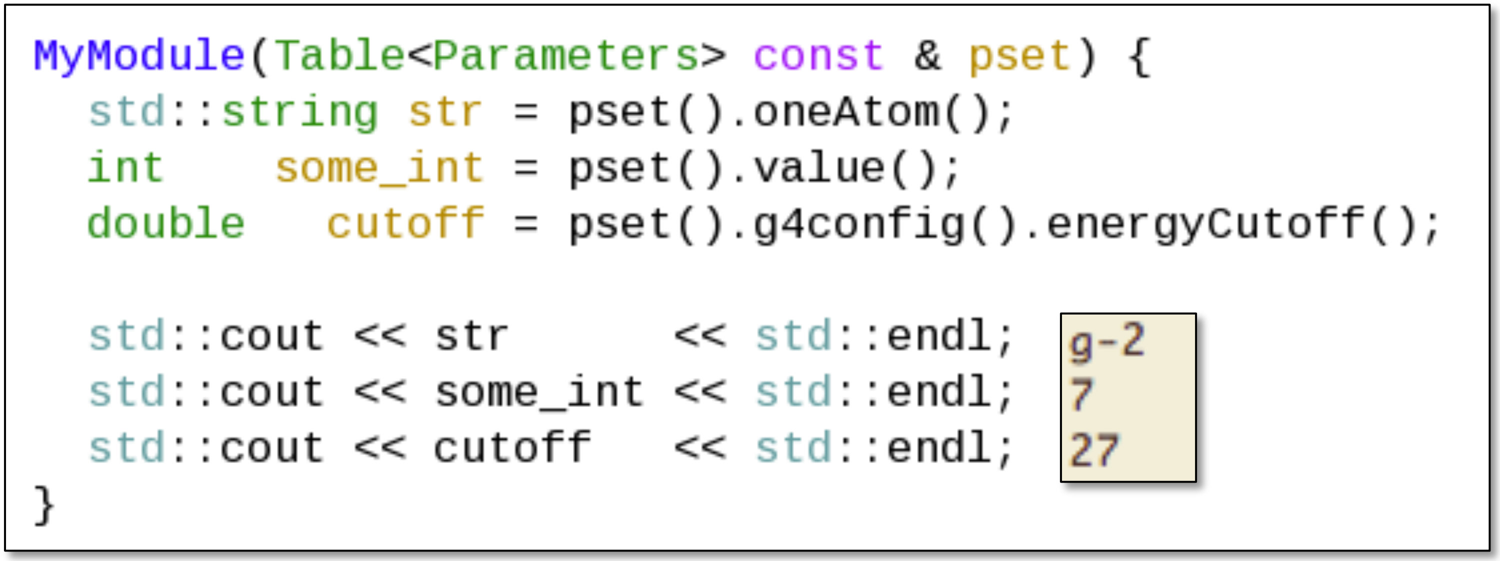
```
MyModule(Table<Parameters> const & pset) {
  std::string str = pset().oneAtom();
  int     some_int = pset().value();
  double    cutoff = pset().g4config().energyCutoff();




}
```

Accessing the elements

- No more `pset.get<>`.
  - Accessing elements is now done using syntax very similar to FHiCL syntax.



```
pset: {

   oneAtom : "g-2"
   value   : 7
   list    : [1,3,17]

   g4Config : {
      energyCutoff : 27
   }

}
```

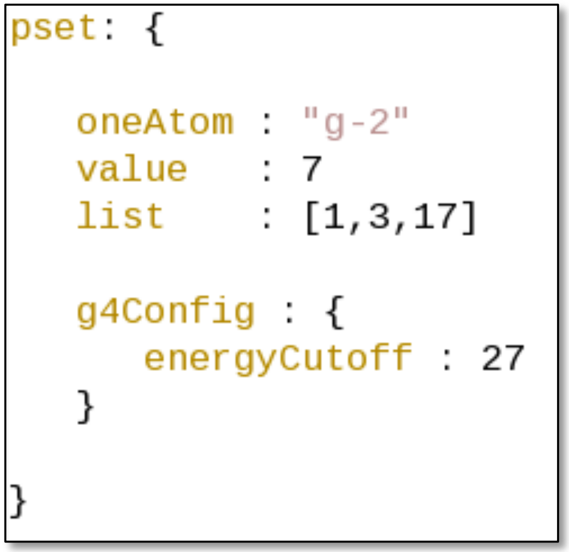


```
MyModule(Table<Parameters> const & pset) {
  std::string str = pset().oneAtom();
  int      some_int = pset().value();
  double    cutoff = pset().g4config().energyCutoff();

  std::cout << str      << std::endl;
  std::cout << some_int << std::endl;
  std::cout << cutoff   << std::endl;
}
```

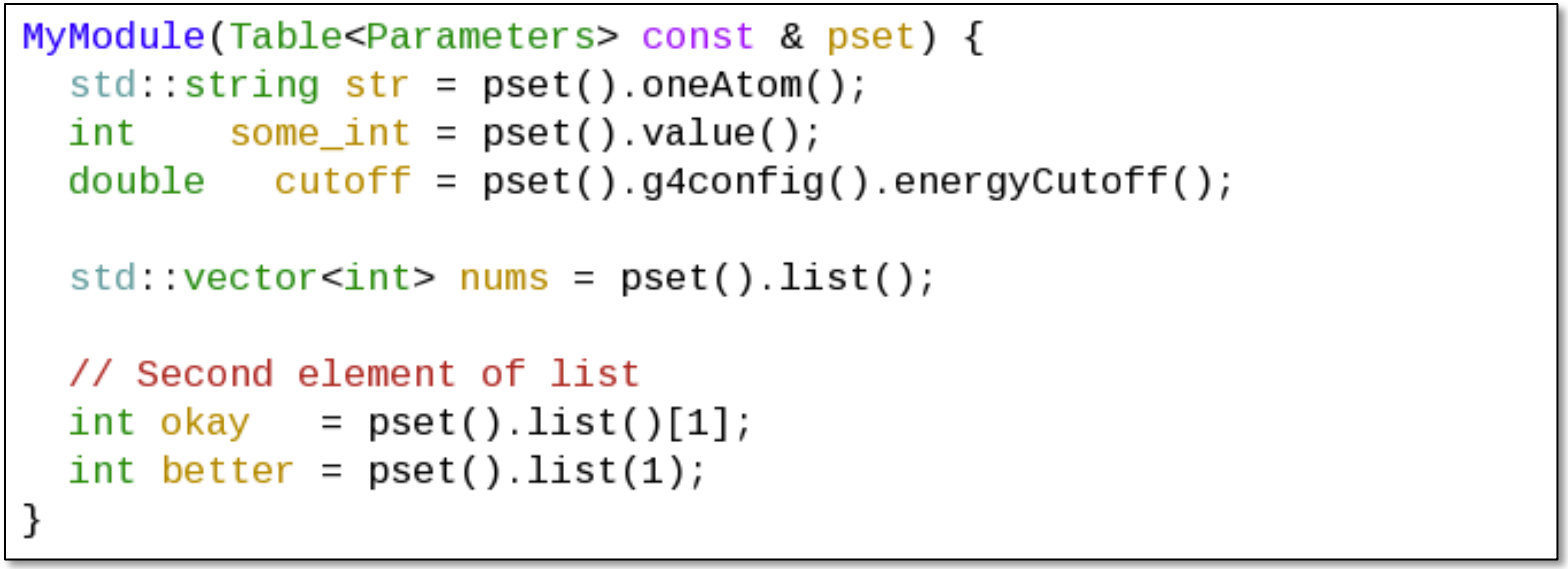
```
g-2
7
27
```

What about sequences?

- No more `pset.get<>`.
  - Accessing elements is now done using syntax very similar to FHiCL syntax.



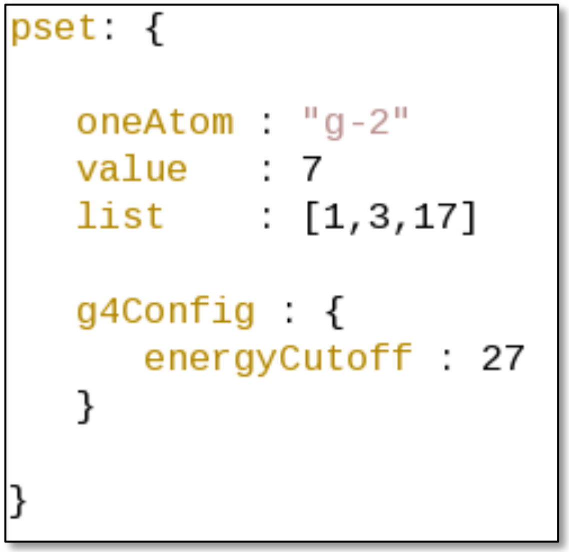
```
pset: {

   oneAtom : "g-2"
   value   : 7
   list    : [1,3,17]

   g4Config : {
      energyCutoff : 27
   }

}
```



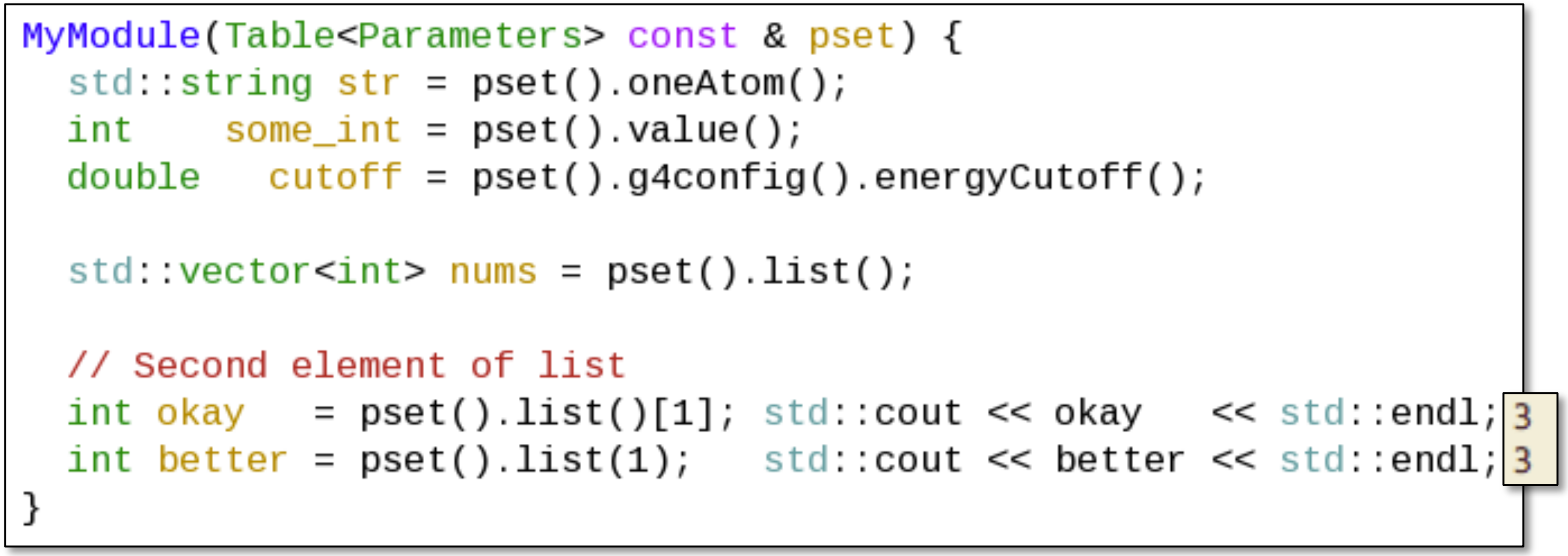
```
MyModule(Table<Parameters> const & pset) {
  std::string str = pset().oneAtom();
  int      some_int = pset().value();
  double     cutoff = pset().g4config().energyCutoff();

  std::vector<int> nums = pset().list();

  // Second element of list
  int okay    = pset().list()[1];
  int better = pset().list(1);
}
```

What about sequences?

- No more `pset.get<>`.
  - Accessing elements is now done using syntax very similar to FHiCL syntax.



```
pset: {

   oneAtom : "g-2"
   value   : 7
   list    : [1,3,17]

   g4Config : {
      energyCutoff : 27
   }

}
```



```
MyModule(Table<Parameters> const & pset) {
  std::string str = pset().oneAtom();
  int     some_int = pset().value();
  double    cutoff = pset().g4config().energyCutoff();

  std::vector<int> nums = pset().list();

  // Second element of list
  int okay    = pset().list()[1]; std::cout << okay   << std::endl; 3
  int better = pset().list(1);    std::cout << better << std::endl; 3
}
```

Features not yet implemented

- Specialized conversions
  - Sometimes you want to read in (e.g.) a sequence as a different kind of structure:

```
pset.get<CLHEP::HepLorentzVector>("lvec");
```

  - Takes 4-element sequence of doubles.

- Will implement specialized versions (e.g.):

```
SequenceAs<CLHEP::HepLorentzVector,double,4>
TupleAs<SomeClass,double,string>
```

Features not yet implemented

- Conditional configuration (e.g.)

```
std::string const& shape = pset.get<string>(“shape”);

if ( shape == “box”) {
  makeBox( pset.get< array<double,3> >(“halfLengths”) );
{
else if (shape == “sphere”) {
  makeSphere( pset.get<double>(“radius”) );
}
```

- Are working to implement something to support this. May not be in place by Aug. 1.

Please give us your input!

- To be released Aug. 1, 2015.
- This is meant to be a help for users of *art*.
- We've made significant progress, but we want your thoughts.
  - knoepfel@fnal.gov , or
  - artists@fnal.gov
- Thanks!