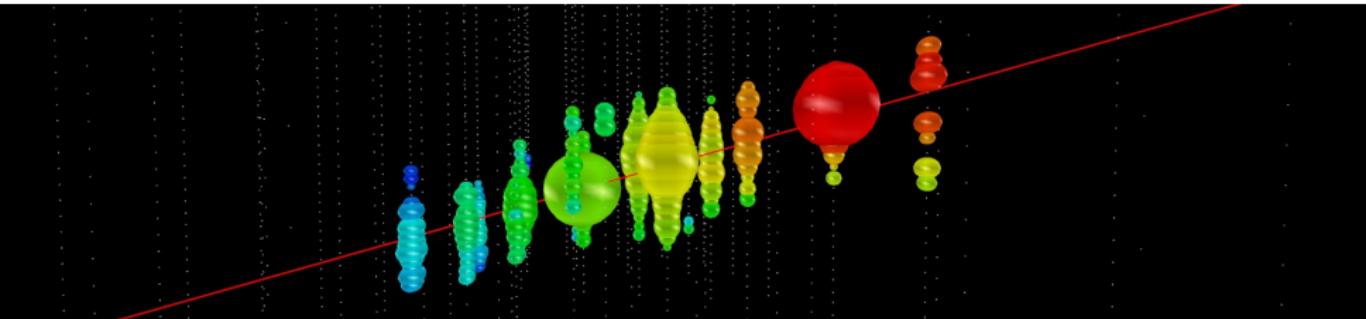


Icefishing for Neutrinos (with boost)

Troy D. Straszheim

Department of Physics
University of Maryland
College Park, MD



Outline

`IceCube`

- `The physics`
- `The detector`

`IceTray`

`boost::serialization`

- `Lazy deserialization`

`boost::python`

`architectural python`

Outline

IceCube

The physics

The detector

IceTray

`boost::serialization`

Lazy deserialization

`boost::python`

architectural python

Neutrinos

ν

- ▶ Nearly as fast as light.
- ▶ No electric charge.
- ▶ Minuscule but nonzero mass.
- ▶ Weakly interacting. Pass right through you.
- ▶ 10^{13} solar electron neutrinos just did.

Neutrinos

ν

- ▶ Nearly as fast as light.
- ▶ No electric charge.
- ▶ Minuscule but nonzero mass.
- ▶ Weakly interacting. Pass right through you.
- ▶ 10^{13} solar electron neutrinos just did.

Neutrinos

ν

- ▶ Nearly as fast as light.
- ▶ No electric charge.
- ▶ Minuscule but nonzero mass.
- ▶ Weakly interacting. Pass right through you.
- ▶ 10^{13} solar electron neutrinos just did.

Neutrinos

ν

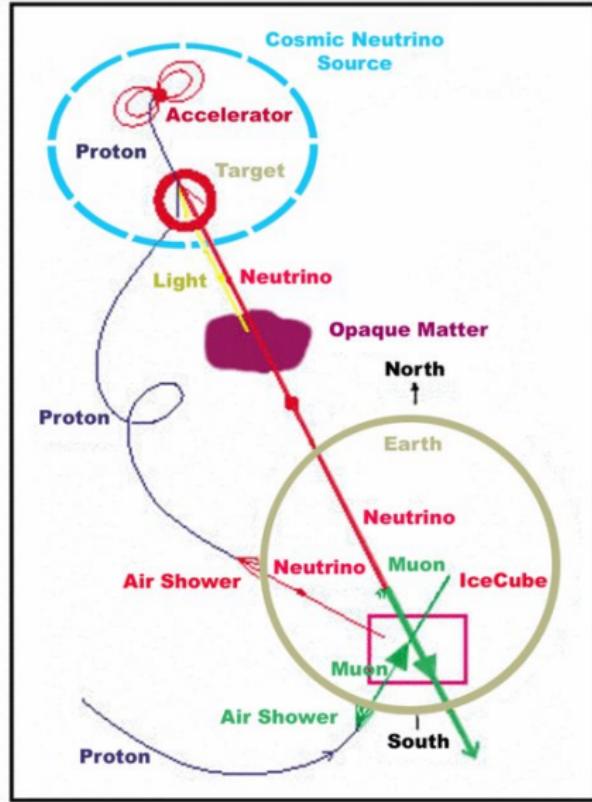
- ▶ Nearly as fast as light.
- ▶ No electric charge.
- ▶ Minuscule but nonzero mass.
- ▶ Weakly interacting. Pass right through you.
- ▶ 10^{13} solar electron neutrinos just did.

Neutrinos

ν

- ▶ Nearly as fast as light.
- ▶ No electric charge.
- ▶ Minuscule but nonzero mass.
- ▶ Weakly interacting. Pass right through you.
- ▶ 10^{13} solar electron neutrinos just did.

Why neutrinos



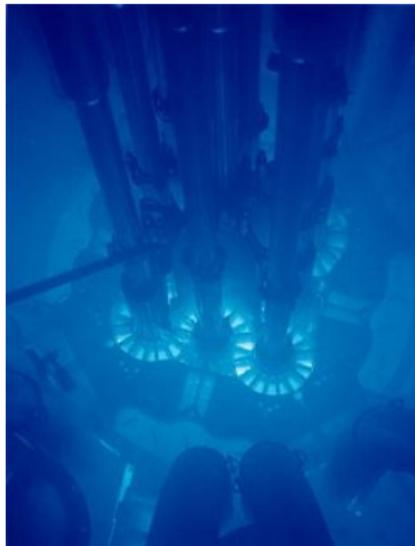
Muons

$$\nu_\mu$$

- ▶ 200x heavier than an electron, 1/10th as heavy as a proton
- ▶ Created when a neutrino collides with a nucleon
- ▶ Orders of magnitude more energy than mass
- ▶ They make...

Cherenkov Radiation

Electromagnetic radiation emitted when a charged particle (such as an electron) passes through an insulator at a speed greater than the phase speed of light in that medium.



Outline

IceCube

The physics

The detector

IceTray

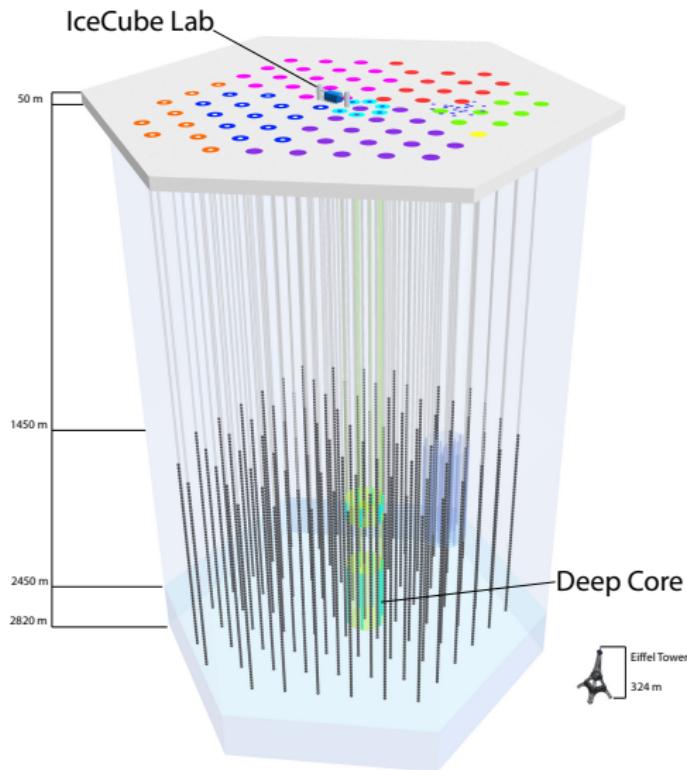
`boost::serialization`

Lazy deserialization

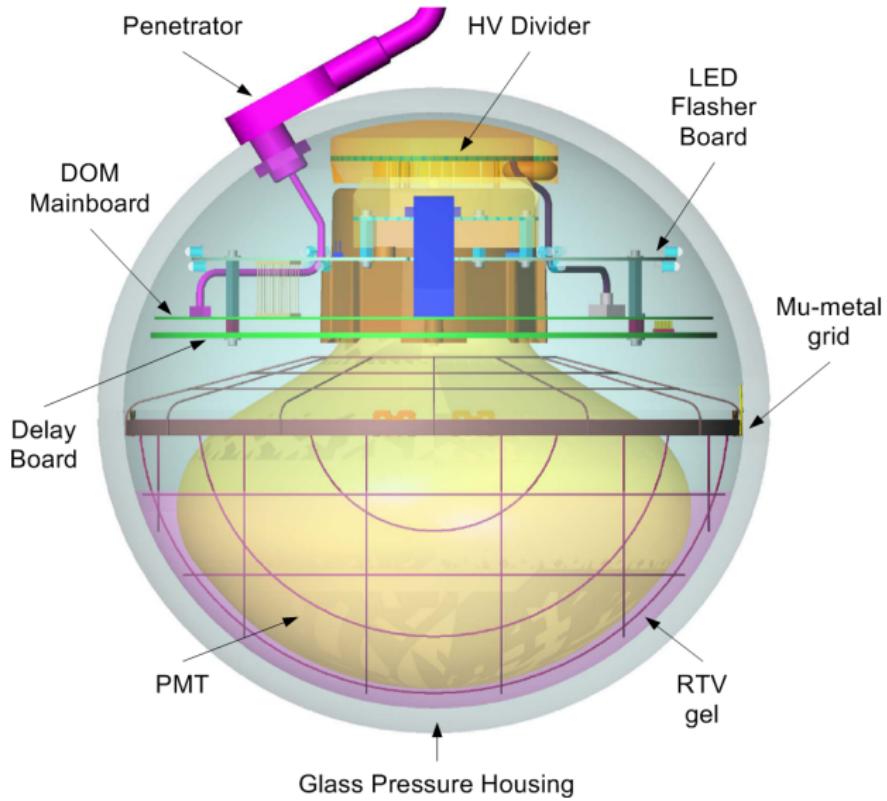
`boost::python`

architectural python

IceCube, a kilometer-scale neutrino detector



Digital Optical Modules



Optical modules ready to deploy

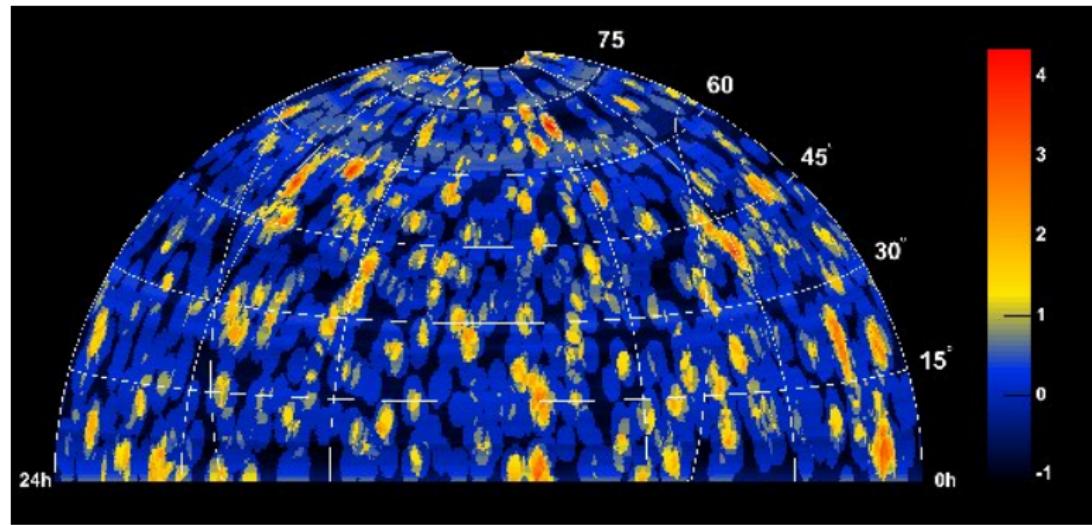


glshovel

drill_high.flv

icecube2-newrender.avi

What we see (nothing, really)



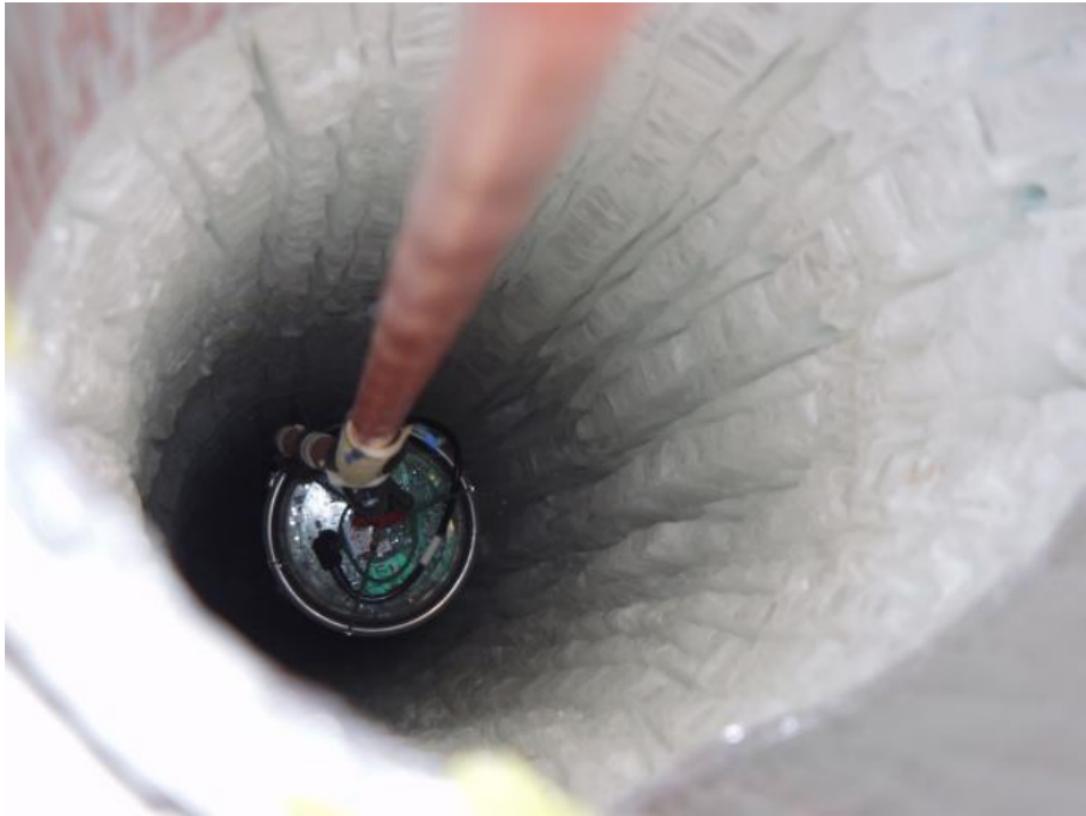
South Pole Station from the air



Powering the drill



DOM deployment



C-130... on skis



More C-130s



C17 with Cargo and Beakers

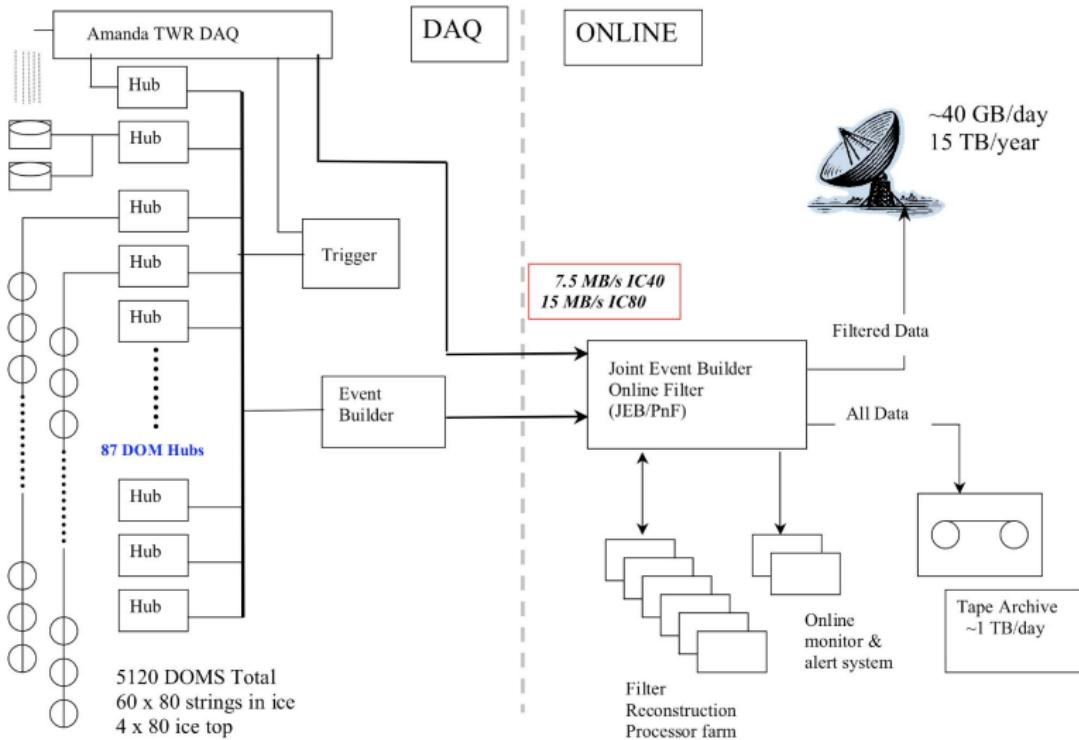


An air drop



(pictures from John Jacobsen, NPX Designs Inc.)

Systems at the South Pole

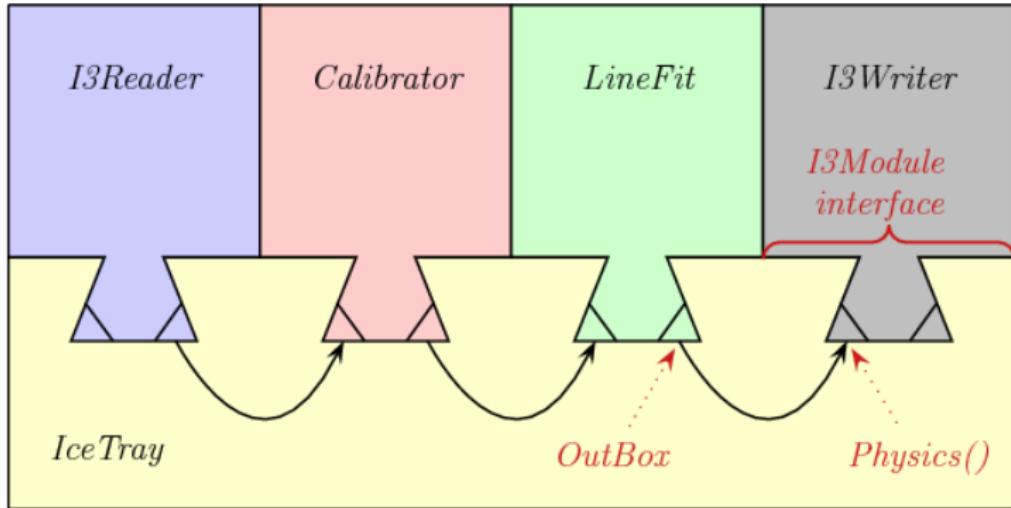


Roles of the IceTray framework

- ▶ Filtering, On Pole
- ▶ Reconstruction
- ▶ Simulation
- ▶ Analysis

- ▶ Core is in C++, as small and simple as possible
- ▶ Scriptable in python

An IceTray instance



A run script

```
from icecube import icetray, dataio, linefit

tray = icetray.I3Tray()

tray.AddModule("I3Reader", "reader",
               Filename = "input.i3.gz")

tray.AddModule("Calibrator", "calibrator")

tray.AddModule("I3Linefit", "linefit",
               MinHits = 24,
               Name = "Linefit_Result")

tray.AddModule("I3Writer", "writer",
               Filename = "output.i3.gz")

tray.Execute()
```

A run script

```
from icecube import icetray, dataio, linefit

tray = icetray.I3Tray()

tray.AddModule("I3Reader", "reader",
               Filename = "input.i3.gz")

tray.AddModule("Calibrator", "calibrator")

tray.AddModule("I3Linefit", "linefit",
               MinHits = 24,
               Name = "Linefit_Result")

tray.AddModule("I3Writer", "writer",
               Filename = "output.i3.gz")

tray.Execute()
```

A run script

```
from icecube import icetray, dataio, linefit

tray = icetray.I3Tray()

tray.AddModule("I3Reader", "reader",
               Filename = "input.i3.gz")

tray.AddModule("Calibrator", "calibrator")

tray.AddModule("I3Linefit", "linefit",
               MinHits = 24,
               Name = "Linefit_Result")

tray.AddModule("I3Writer", "writer",
               Filename = "output.i3.gz")

tray.Execute()
```

A run script

```
from icecube import icetray, dataio, linefit

tray = icetray.I3Tray()

tray.AddModule("I3Reader", "reader",
               Filename = "input.i3.gz")

tray.AddModule("Calibrator", "calibrator")

tray.AddModule("I3Linefit", "linefit",
               MinHits = 24,
               Name = "Linefit_Result")

tray.AddModule("I3Writer", "writer",
               Filename = "output.i3.gz")

tray.Execute()
```

A run script

```
from icecube import icetray, dataio, linefit

tray = icetray.I3Tray()

tray.AddModule("I3Reader", "reader",
               Filename = "input.i3.gz")

tray.AddModule("Calibrator", "calibrator")

tray.AddModule("I3Linefit", "linefit",
               MinHits = 24,
               Name = "Linefit_Result")

tray.AddModule("I3Writer", "writer",
               Filename = "output.i3.gz")

tray.Execute()
```

A run script

```
from icecube import icetray, dataio, linefit

tray = icetray.I3Tray()

tray.AddModule("I3Reader", "reader",
               Filename = "input.i3.gz")

tray.AddModule("Calibrator", "calibrator")

tray.AddModule("I3Linefit", "linefit",
               MinHits = 24,
               Name = "Linefit_Result")

tray.AddModule("I3Writer", "writer",
               Filename = "output.i3.gz")

tray.Execute()
```

A run script

```
from icecube import icetray, dataio, linefit

tray = icetray.I3Tray()

tray.AddModule("I3Reader", "reader",
               Filename = "input.i3.gz")

tray.AddModule("Calibrator", "calibrator")

tray.AddModule("I3Linefit", "linefit",
               MinHits = 24,
               Name = "Linefit_Result")

tray.AddModule("I3Writer", "writer",
               Filename = "output.i3.gz")

tray.Execute()
```

A run script

```
from icecube import icetray, dataio, linefit

tray = icetray.I3Tray()

tray.AddModule("I3Reader", "reader",
               Filename = "input.i3.gz")

tray.AddModule("Calibrator", "calibrator")

tray.AddModule("I3Linefit", "linefit",
               MinHits = 24,
               Name = "Linefit_Result")

tray.AddModule("I3Writer", "writer",
               Filename = "output.i3.gz")

tray.Execute()
```

A run script

```
from icecube import icetray, dataio, linefit

tray = icetray.I3Tray()

tray.AddModule("I3Reader", "reader",
               Filename = "input.i3.gz")
tray.SetParameter("reader", "Filename", "input.i3.gz")
tray.AddModule("Calibrator", "calibrator")

tray.AddModule("I3Linefit", "linefit",
               MinHits = 24,
               Name = "Linefit_Result")

tray.AddModule("I3Writer", "writer",
               Filename = "output.i3.gz")

tray.Execute()
```

A run script

```
from icecube import icetray, dataio, linefit

tray = icetray.I3Tray()

tray.AddModule("I3Reader", "reader",
               Filename = "input.i3.gz")
tray.SetParameter("reader", "Filename", "input.i3.gz")
tray.AddModule("Calibrator", "calibrator")

tray.AddModule("I3Linefit", "linefit",
               MinHits = 24,
               Name = "Linefit_Result")

tray.AddModule("I3Writer", "writer",
               Filename = "output.i3.gz")

tray.Execute()
```

A run script

```
from icecube import icetray, dataio, linefit

tray = icetray.I3Tray()

tray.AddModule("I3Reader", "reader",
               Filename = "input.i3.gz")
tray.SetParameter("reader", "Filename", "input.i3.gz")
tray.AddModule("Calibrator", "calibrator")

tray.AddModule("I3Linefit", "linefit",
               MinHits = 24,
               Name = "Linefit_Result")

tray.AddModule("I3Writer", "writer",
               Filename = "output.i3.gz")

tray.Execute()
```

A run script

```
from icecube import icetray, dataio, linefit

tray = icetray.I3Tray()

tray.AddModule("I3Reader", "reader",
               Filename = "input.i3.gz")

tray.AddModule("Calibrator", "calibrator")

tray.AddModule("I3Linefit", "linefit",
                 MinHits = 24,
                 Name = "Linefit_Result")

tray.AddModule("I3Writer", "writer",
                 Filename = "output.i3.gz")

tray.Execute()
```

A run script

```
from icecube import icetray, dataio, linefit

tray = icetray.I3Tray()

tray.AddModule("I3Reader", "reader",
               Filename = "input.i3.gz")

tray.AddModule("Calibrator", "calibrator")

tray.AddModule("I3Linefit", "linefit",
               MinHits = 24,
               Name = "Linefit_Result")

tray.AddModule("I3Writer", "writer",
               Filename = "output.i3.gz")

tray.Execute()
```

The lifecycle of a module

- ▶ Framework loads shared libraries containing modules
- ▶ User requests that framework create a module `Foo` named `foo_inst`
- ▶ Framework creates `foo_inst`, which reports its parameter space
- ▶ Framework connects the *outboxes* and *inboxes* of various modules
- ▶ Framework calls `void Configure()`, wherein module gets the user-configured values of its parameters
- ▶ Framework repeatedly calls `void Physics(I3FramePtr frame)`

The lifecycle of a module

- ▶ Framework loads shared libraries containing modules
- ▶ User requests that framework create a module `Foo` named `foo_inst`
- ▶ Framework creates `foo_inst`, which reports its parameter space
- ▶ Framework connects the *outboxes* and *inboxes* of various modules
- ▶ Framework calls `void Configure()`, wherein module gets the user-configured values of its parameters
- ▶ Framework repeatedly calls `void Physics(I3FramePtr frame)`

The lifecycle of a module

- ▶ Framework loads shared libraries containing modules
- ▶ User requests that framework create a module `Foo` named `foo_inst`
- ▶ Framework creates `foo_inst`, which reports its parameter space
- ▶ Framework connects the *outboxes* and *inboxes* of various modules
- ▶ Framework calls `void Configure()`, wherein module gets the user-configured values of its parameters
- ▶ Framework repeatedly calls `void Physics(I3FramePtr frame)`

The lifecycle of a module

- ▶ Framework loads shared libraries containing modules
- ▶ User requests that framework create a module `Foo` named `foo_inst`
- ▶ Framework creates `foo_inst`, which reports its parameter space
- ▶ Framework connects the *outboxes* and *inboxes* of various modules
- ▶ Framework calls `void Configure()`, wherein module gets the user-configured values of its parameters
- ▶ Framework repeatedly calls `void Physics(I3FramePtr frame)`

The lifecycle of a module

- ▶ Framework loads shared libraries containing modules
- ▶ User requests that framework create a module `Foo` named `foo_inst`
- ▶ Framework creates `foo_inst`, which reports its parameter space
- ▶ Framework connects the *outboxes* and *inboxes* of various modules
- ▶ Framework calls `void Configure()`, wherein module gets the user-configured values of its parameters
- ▶ Framework repeatedly calls `void Physics(I3FramePtr frame)`

The lifecycle of a module

- ▶ Framework loads shared libraries containing modules
- ▶ User requests that framework create a module `Foo` named `foo_inst`
- ▶ Framework creates `foo_inst`, which reports its parameter space
- ▶ Framework connects the *outboxes* and *inboxes* of various modules
- ▶ Framework calls `void Configure()`, wherein module gets the user-configured values of its parameters
- ▶ Framework repeatedly calls `void Physics(I3FramePtr frame)`

Automating pointer typedefs

```
#define I3_POINTER_TYPEDEFS(C) \
    typedef boost::shared_ptr<C> C##Ptr; \
    typedef boost::shared_ptr<const C> C##ConstPtr; \
 \
class MyClass { ... }; \
 \
I3_POINTER_TYPEDEFS(MyClass); \
 \
// gives you MyClassPtr and MyClassConstPtr
```

The I3Module interface (simplified)

```
class I3Module
{
public:
    I3Module(const I3Context&);
    virtual void Configure();
    virtual void Physics(I3FramePtr);

protected:

    template <typename T>
    void AddParameter(string name, T default);

    template <typename T>
    void GetParameter(string name, T& value);

    void PushFrame(I3FramePtr);

private:
    const I3Context& context_;
};
```

The I3Module interface (simplified)

```
class I3Module
{
public:
    I3Module(const I3Context&); // map<string,boost::any>
    virtual void Configure();
    virtual void Physics(I3FramePtr);

protected:

    template <typename T>
    void AddParameter(string name, T default);

    template <typename T>
    void GetParameter(string name, T& value);

    void PushFrame(I3FramePtr);

private:
    const I3Context& context_;
};
```

The I3Module interface (simplified)

```
class I3Module
{
public:
    I3Module(const I3Context&);

virtual void Configure();

    virtual void Physics(I3FramePtr);

protected:

    template <typename T>
    void AddParameter(string name, T default_);

    template <typename T>
    void GetParameter(string name, T& value);

    void PushFrame(I3FramePtr);

private:
    const I3Context& context_;
};
```

The I3Module interface (simplified)

```
class I3Module
{
public:
    I3Module(const I3Context&);

    virtual void Configure();
    virtual void Physics(I3FramePtr);

protected:

    template <typename T>
    void AddParameter(string name, T default);

    template <typename T>
    void GetParameter(string name, T& value);

    void PushFrame(I3FramePtr);

private:
    const I3Context& context_;
};
```

The I3Module interface (simplified)

```
class I3Module
{
public:
    I3Module(const I3Context&);

    virtual void Configure();
    virtual void Physics(I3FramePtr);

protected:

    template <typename T>
    void AddParameter(string name, T default_);

    template <typename T>
    void GetParameter(string name, T& value);

    void PushFrame(I3FramePtr);

private:
    const I3Context& context_;
};
```

The I3Module interface (simplified)

```
class I3Module
{
public:
    I3Module(const I3Context&);

    virtual void Configure();
    virtual void Physics(I3FramePtr);

protected:

    template <typename T>
    void AddParameter(string name, T default_);

    template <typename T>
    void GetParameter(string name, T& value);

    void PushFrame(I3FramePtr);

private:
    const I3Context& context_;
};

};
```

The I3Module interface (simplified)

```
class I3Module
{
public:
    I3Module(const I3Context&);
    virtual void Configure();
    virtual void Physics(I3FramePtr);

protected:

    template <typename T>
    void AddParameter(string name, T default);

    template <typename T>
    void GetParameter(string name, T& value);

    void PushFrame(I3FramePtr);

private:
    const I3Context& context_;
};
```

The I3Module interface (simplified)

```
class I3Module
{
public:
    I3Module(const I3Context&);

    virtual void Configure();
    virtual void Physics(I3FramePtr);

protected:

    template <typename T>
    void AddParameter(string name, T default);

    template <typename T>
    void GetParameter(string name, T& value);

    void PushFrame(I3FramePtr);

private:
    const I3Context& context_;
};
```

A simple module

```
class MyFilterModule : public I3Module
{
    std::string key;
public:
    MyFilterModule(const I3Context& c) : I3Module(c)
    {
        AddParameter("key", "(nil)");
    }
    void Configure()
    {
        GetParameter("key", key);
    }
    void Physics(I3FramePtr frame)
    {
        // does something with the frame
        PushFrame(frame);
    }
};

I3_MODULE(MyFilterModule);
```

A simple module

```
class MyFilterModule : public I3Module
{
    std::string key;
public:
    MyFilterModule(const I3Context& c) : I3Module(c)
    {
        AddParameter("key", "(nil)");
    }
    void Configure()
    {
        GetParameter("key", key);
    }
    void Physics(I3FramePtr frame)
    {
        // does something with the frame
        PushFrame(frame);
    }
};

I3_MODULE(MyFilterModule);
```

A simple module

```
class MyFilterModule : public I3Module
{
    std::string key;
public:
    MyFilterModule(const I3Context& c) : I3Module(c)
    {
        AddParameter("key", "(nil)");
    }
void Configure()
{
    GetParameter("key", key);
}
void Physics(I3FramePtr frame)
{
    // does something with the frame
    PushFrame(frame);
}
I3_MODULE(MyFilterModule);
```

A simple module

```
class MyFilterModule : public I3Module
{
    std::string key;
public:
    MyFilterModule(const I3Context& c) : I3Module(c)
    {
        AddParameter("key", "(nil)");
    }
    void Configure()
    {
        GetParameter("key", key);
    }
void Physics(I3FramePtr frame)
{
    // does something with the frame
    PushFrame(frame);
}
};

I3_MODULE(MyFilterModule);
```

A simple module

```
class MyFilterModule : public I3Module
{
    std::string key;
public:
    MyFilterModule(const I3Context& c) : I3Module(c)
    {
        AddParameter("key", "(nil)");
    }
    void Configure()
    {
        GetParameter("key", key);
    }
    void Physics(I3FramePtr frame)
    {
        // does something with the frame
        PushFrame(frame);
    }
};

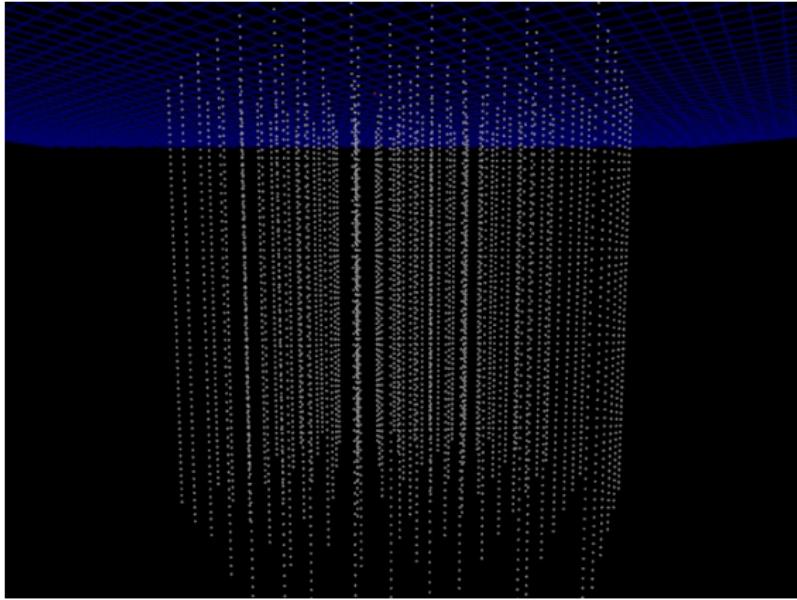
I3_MODULE(MyFilterModule);
```

A simple module

```
class MyFilterModule : public I3Module
{
    std::string key;
public:
    MyFilterModule(const I3Context& c) : I3Module(c)
    {
        AddParameter("key", "(nil)");
    }
    void Configure()
    {
        GetParameter("key", key);
    }
    void Physics(I3FramePtr frame)
    {
        // does something with the frame
        PushFrame(frame);
    }
};

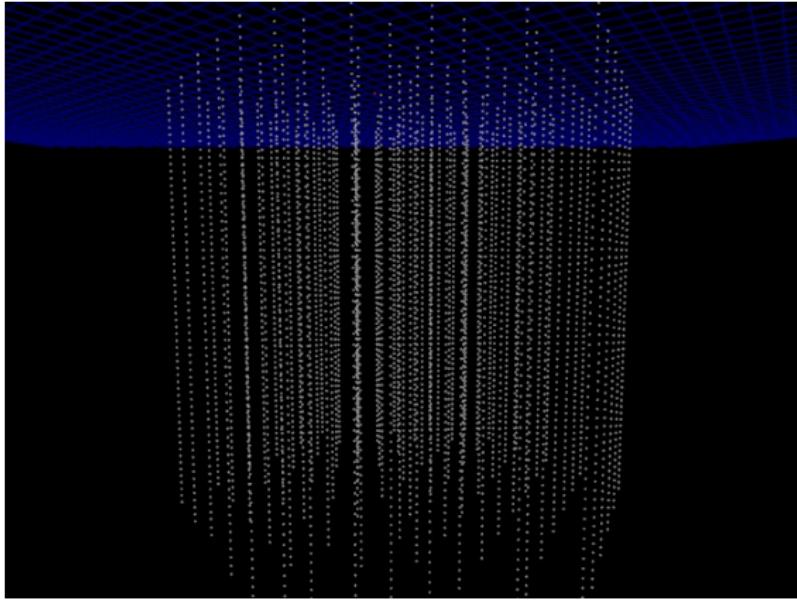
I3_MODULE(MyFilterModule);
```

The “Geometry” of the detector



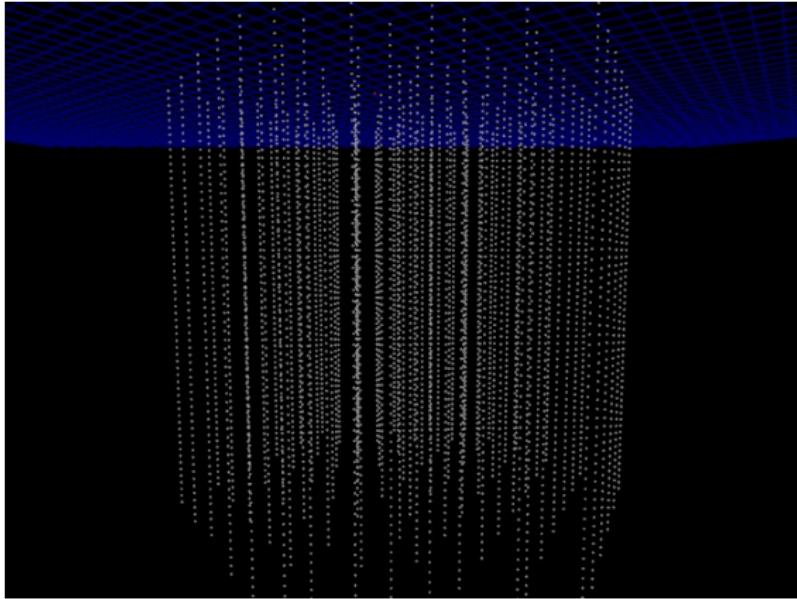
```
struct OMKey { // stringNumber, omNumber...
struct I3OM { // position, om_type, orientation,
              // area, azimuth ...
struct I3Geometry { map<OMKey, I3OM> ...
```

The “Geometry” of the detector



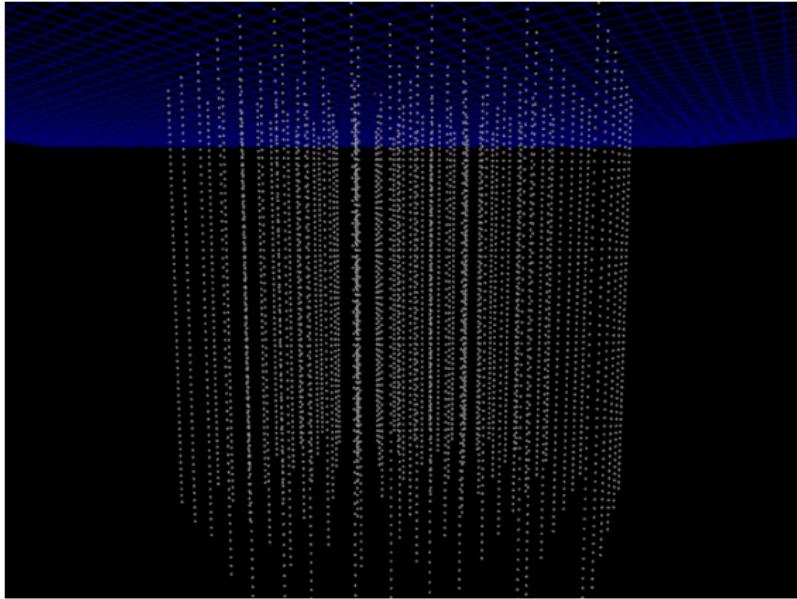
```
struct OMKey { // stringNumber, omNumber...
struct I3OM { // position, om_type, orientation,
              // area, azimuth ...
struct I3Geometry { map<OMKey, I3OM> ...
```

The “Geometry” of the detector



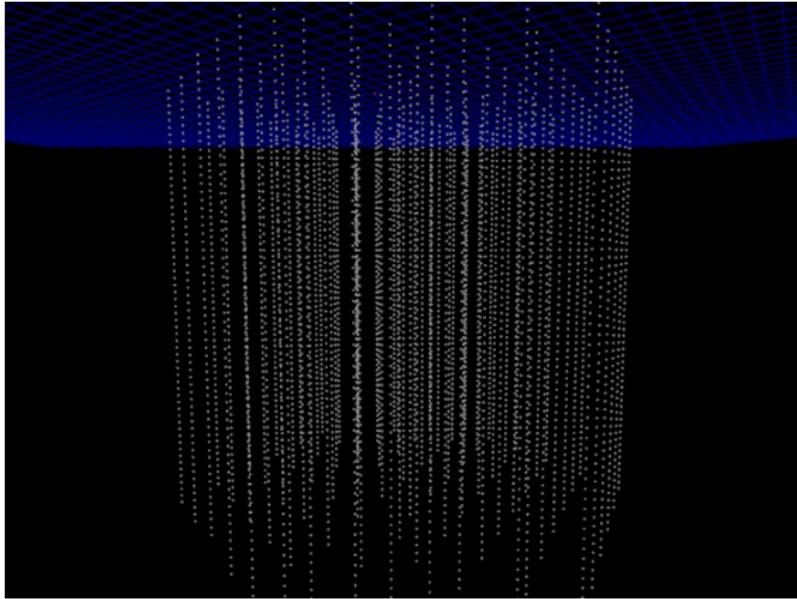
```
struct OMKey { // stringNumber, omNumber...
struct I3OM { // position, om_type, orientation,
    // area, azimuth ...
struct I3Geometry { map<OMKey, I3OM> ...
```

The “Geometry” of the detector



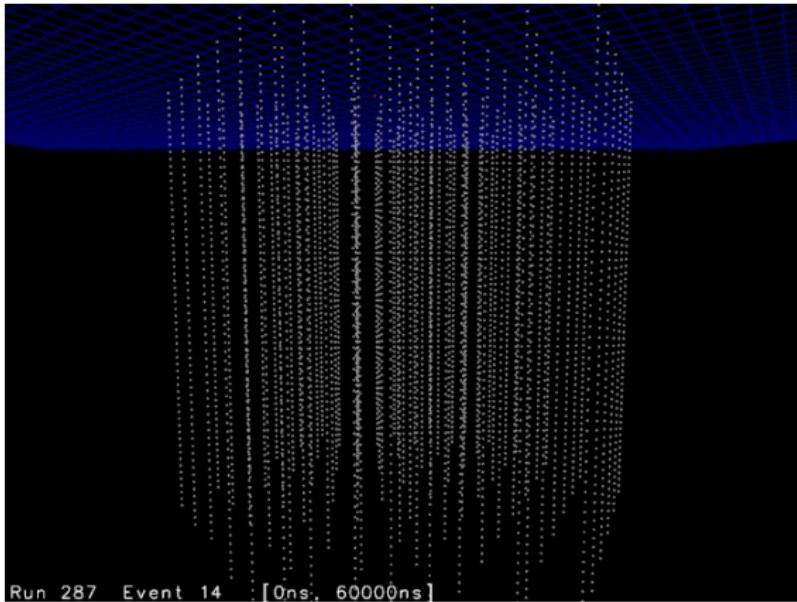
```
struct OMKey { // stringNumber, omNumber...
struct I3OM { // position, om_type, orientation,
              // area, azimuth ...
struct I3Geometry { map<OMKey, I3OM> ...
```

The “Geometry” of the detector



```
struct OMKey { // stringNumber, omNumber...
struct I3OM { // position, om_type, orientation,
              // area, azimuth ...
struct I3Geometry { map<OMKey, I3OM> ...
```

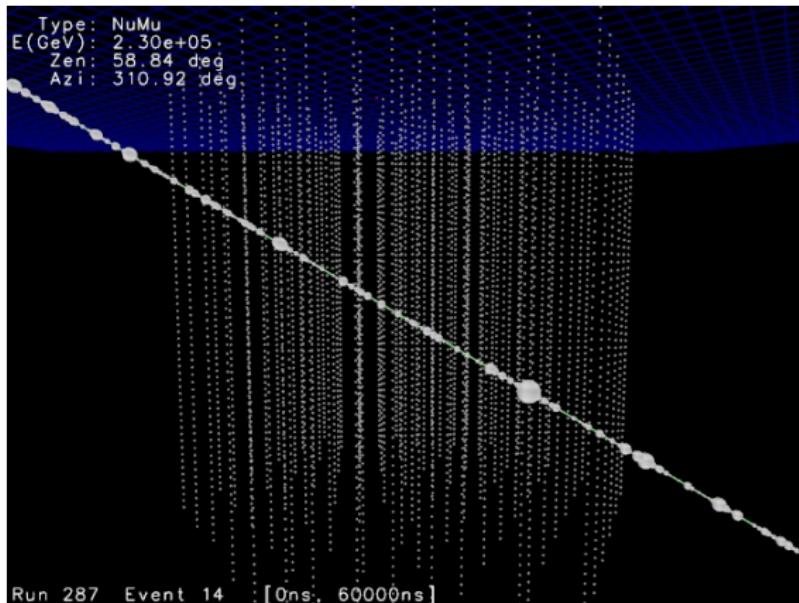
Bookkeeping info about this event



Run 287 Event 14 [0ns, 60000ns]

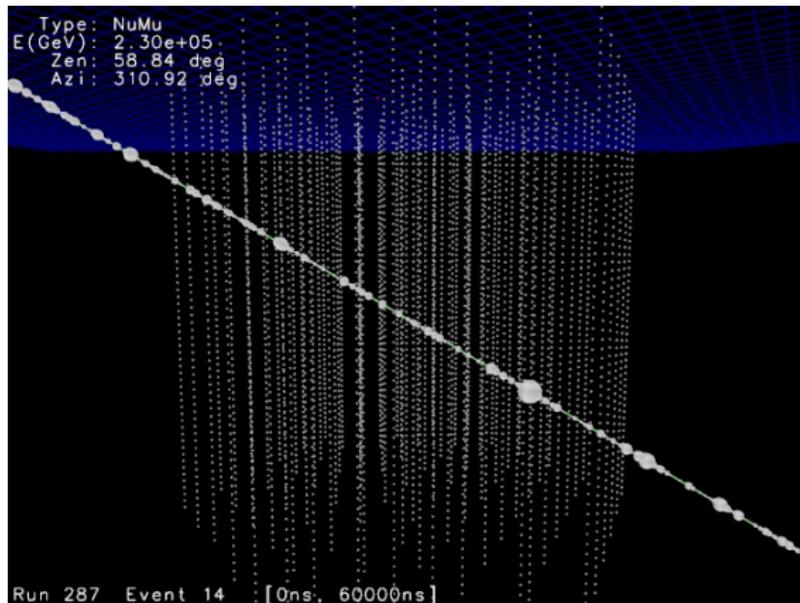
```
struct I3EventHeader {
    // run_id, subrun_id, event_id,
    // state, start_time, end_time
}
```

The monte carlo 'truth': I3Tree<I3Particle>



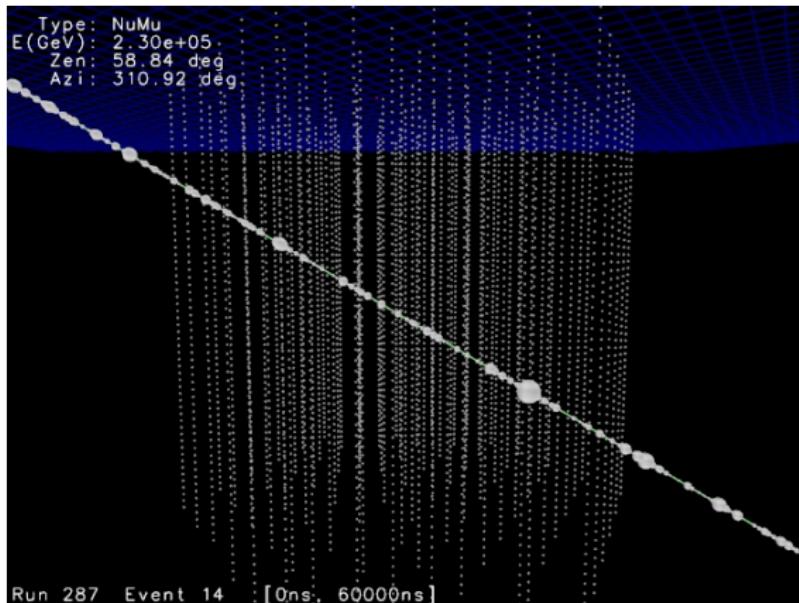
```
struct I3Particle {  
    // position, direction, energy, length, speed, time,  
    // particle_type, particle_shape...  
}
```

The monte carlo 'truth': **I3Tree<I3Particle>**



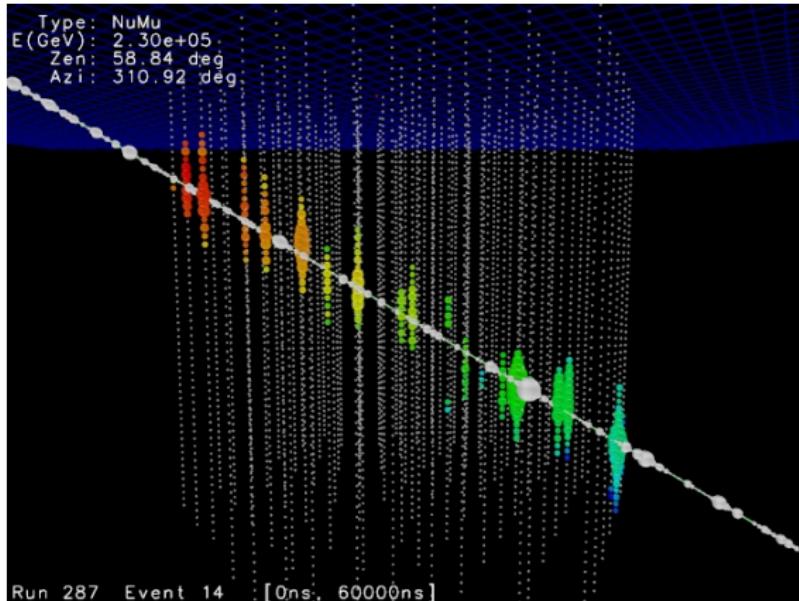
```
struct I3Particle {  
    // position, direction, energy, length, speed, time,  
    // particle_type, particle_shape...  
}
```

The monte carlo 'truth': I3Tree<**I3Particle**>



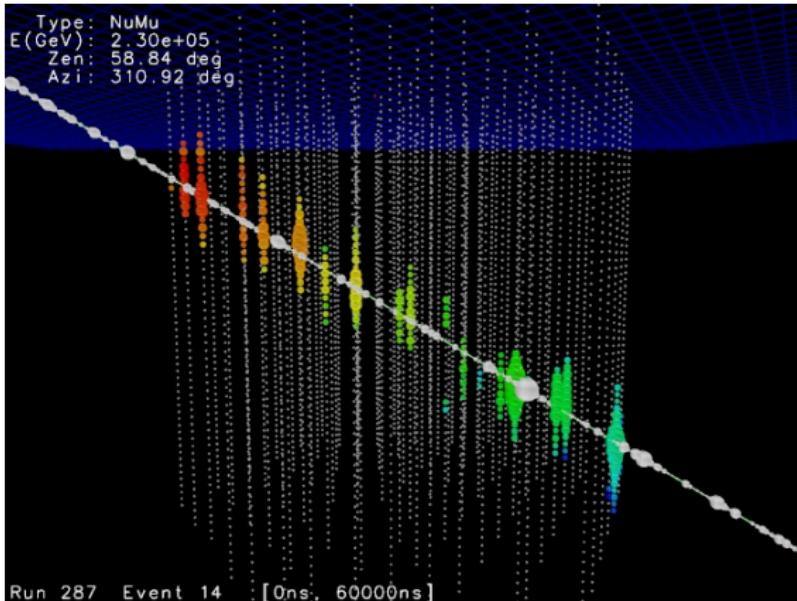
```
struct I3Particle {  
    // position, direction, energy, length, speed, time,  
    // particle_type, particle_shape...  
}
```

Monte carlo part 2: map<OMKey, vector<I3MCHit> >



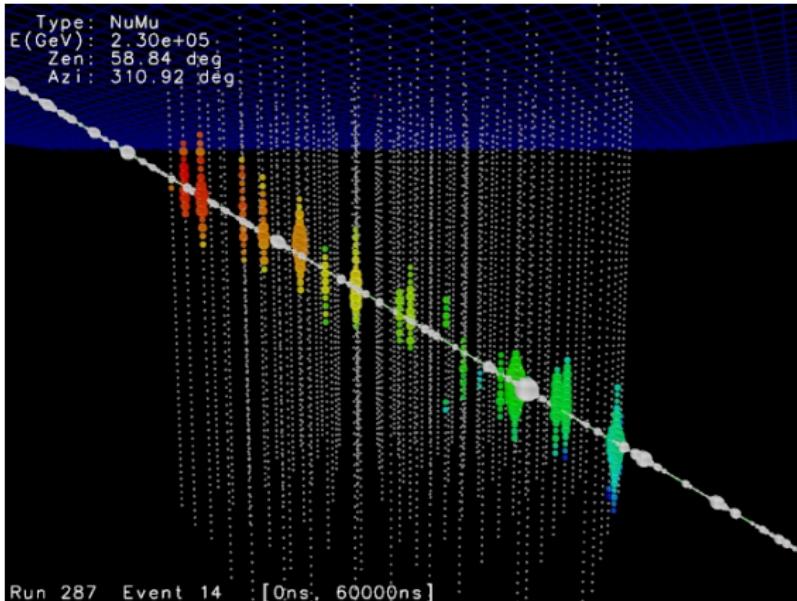
```
struct I3MCHit {  
    // time, weight, source, particle_id  
  
}
```

Monte carlo part 2: **map<OMKey, vector<I3MCHit> >**



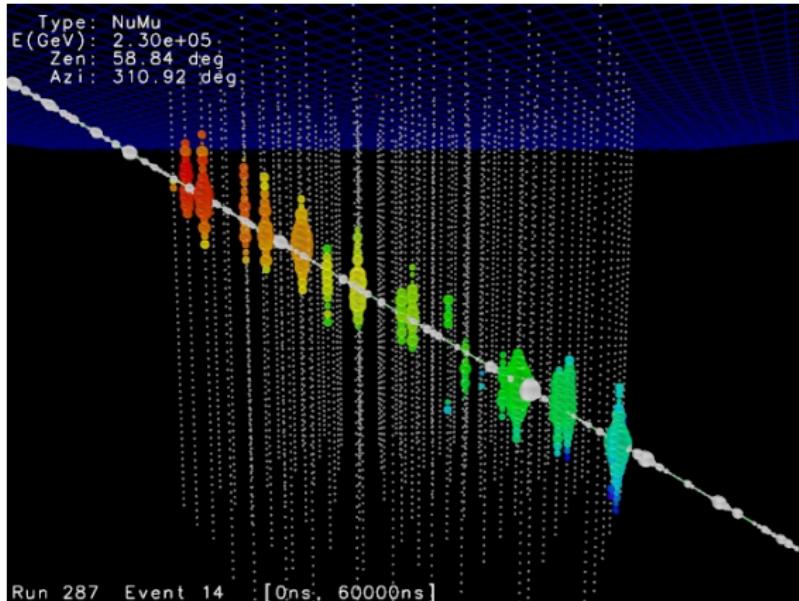
```
struct I3MCHit {  
    // time, weight, source, particle_id  
  
}
```

Monte carlo part 2: map<OMKey, vector<**I3MCHit**> >



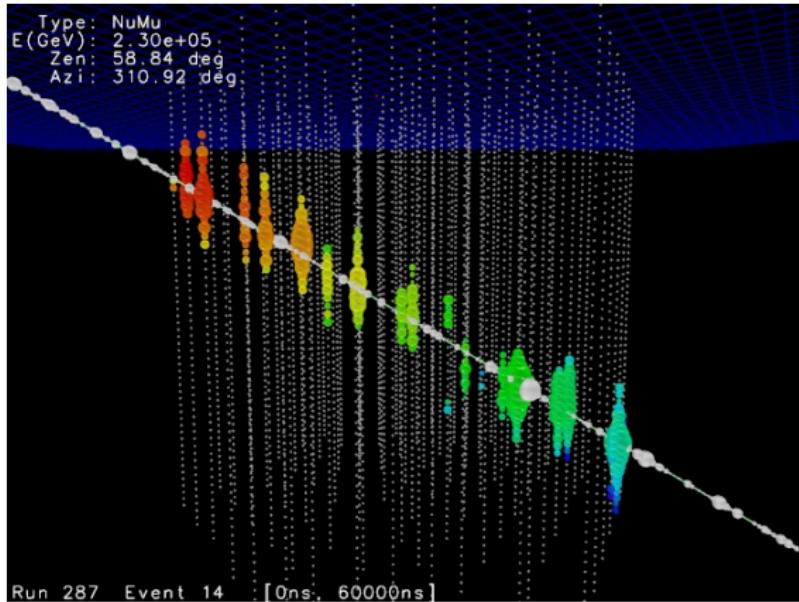
```
struct I3MCHit {  
    // time, weight, source, particle_id  
}
```

Reconstruction part 1: map<OMKey, vector<I3RecoPulse> >



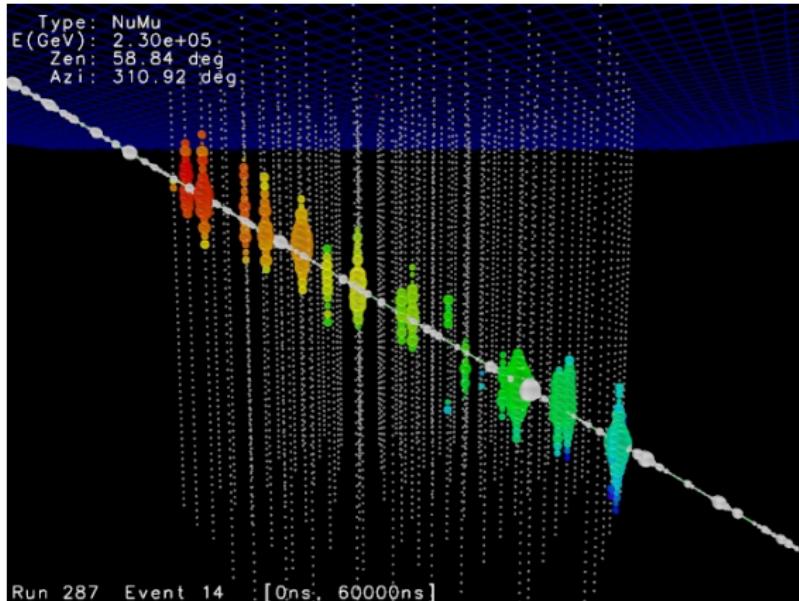
```
struct I3RecoPulse {  
    // time, charge, width, sourceindex  
}
```

Reconstruction part 1: map<OMKey, vector<I3RecoPulse> >



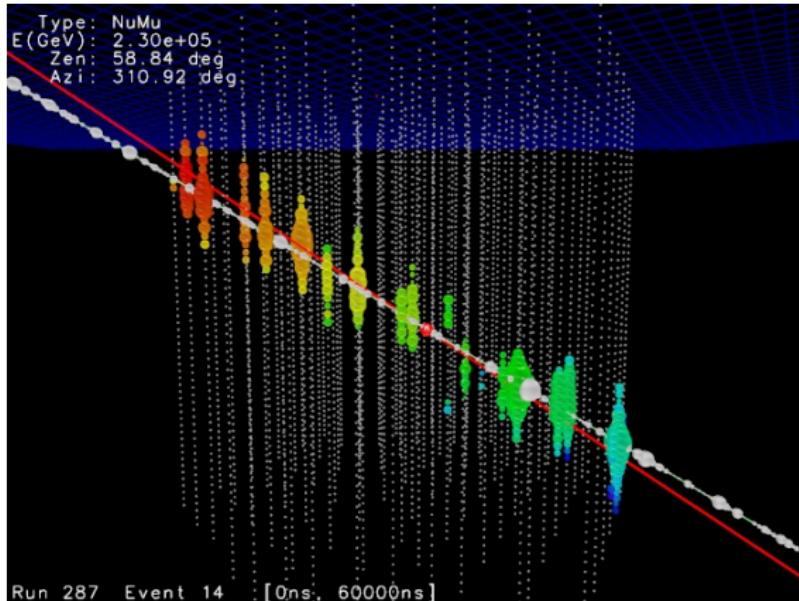
```
struct I3RecoPulse {  
    // time, charge, width, sourceindex  
}
```

Reconstruction part 1: map<OMKey, vector<**I3RecoPulse**> >



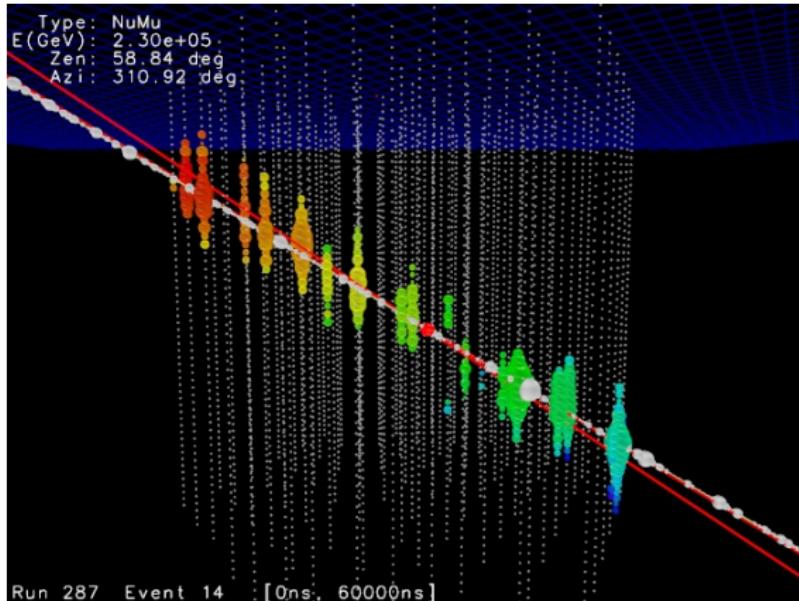
```
struct I3RecoPulse {  
    // time, charge, width, sourceindex  
}
```

The goal: reconstructed I3Particle



```
struct I3Particle {  
    // position, direction, energy, length, speed, time,  
    // particle_type, particle_shape...  
}
```

Several reconstructed I3Particles



```
struct I3Particle {  
    // position, direction, energy, length, speed, time,  
    // particle_type, particle_shape...  
}
```

Requirements for the I3Frame

- ▶ We don't know what people are going to want to put in it
- ▶ Users should be able to put stuff into it that we don't know about
- ▶ Should be able to have the same datatype, multiple times (different reconstruction and/or filtering algorithms)
- ▶ Quasi-const. If objects inside the frame get modified, we want to be able to see who did it. We don't want it to be possible to do accidentally.

Requirements for the I3Frame

- ▶ We don't know what people are going to want to put in it
- ▶ Users should be able to put stuff into it that we don't know about
- ▶ Should be able to have the same datatype, multiple times (different reconstruction and/or filtering algorithms)
- ▶ Quasi-const. If objects inside the frame get modified, we want to be able to see who did it. We don't want it to be possible to do accidentally.

Requirements for the I3Frame

- ▶ We don't know what people are going to want to put in it
- ▶ Users should be able to put stuff into it that we don't know about
- ▶ Should be able to have the same datatype, multiple times (different reconstruction and/or filtering algorithms)
- ▶ Quasi-const. If objects inside the frame get modified, we want to be able to see who did it. We don't want it to be possible to do accidentally.

Requirements for the I3Frame

- ▶ We don't know what people are going to want to put in it
- ▶ Users should be able to put stuff into it that we don't know about
- ▶ Should be able to have the same datatype, multiple times (different reconstruction and/or filtering algorithms)
- ▶ Quasi-const. If objects inside the frame get modified, we want to be able to see who did it. We don't want it to be possible to do accidentally.

Requirements for the I3Frame

- ▶ We don't know what people are going to want to put in it
- ▶ Users should be able to put stuff into it that we don't know about
- ▶ Should be able to have the same datatype, multiple times (different reconstruction and/or filtering algorithms)
- ▶ Quasi-const. If objects inside the frame get modified, we want to be able to see who did it. We don't want it to be possible to do accidentally.

Result: Implement as `map<string, pointer-to-base>`

The I3Frame

```
struct I3FrameObject {
    virtual ~I3FrameObject () ;
};

struct I3EventHeader : I3FrameObject {
    int run_id, subrun_id, event_id, etc;
};

typedef I3Frame map<string, I3FrameObjectPtr>;

void MyModule::Physics(I3FramePtr frame)
{
    I3FrameObjectPtr fop = (*frame)['hitseries'];
    if (!fop) throw_exception(...);
    I3EventHeaderPtr eh =
        shared_pointer_cast<I3EventHeader>(fop);
    if (!eh) throw_exception(...);
    ....
```

The I3Frame

```
struct I3FrameObject {
    virtual ~I3FrameObject () ;
};

struct I3EventHeader : I3FrameObject {
    int run_id, subrun_id, event_id, etc;
};

typedef I3Frame map<string, I3FrameObjectPtr>;

void MyModule::Physics(I3FramePtr frame)
{
    I3FrameObjectPtr fop = (*frame)['hitseries'];
    if (!fop) throw_exception(...);
    I3EventHeaderPtr eh =
        shared_pointer_cast<I3EventHeader>(fop);
    if (!eh) throw_exception(...);
    ....
```

The I3Frame

```
struct I3FrameObject {
    virtual ~I3FrameObject () ;
};

struct I3EventHeader : I3FrameObject {
    int run_id, subrun_id, event_id, etc;
};

typedef I3Frame map<string, I3FrameObjectPtr>;

void MyModule::Physics(I3FramePtr frame)
{
    I3FrameObjectPtr fop = (*frame)['hitseries'];
    if (!fop) throw_exception(...);
    I3EventHeaderPtr eh =
        shared_pointer_cast<I3EventHeader>(fop);
    if (!eh) throw_exception(...);
    ....
```

The I3Frame

```
struct I3FrameObject {
    virtual ~I3FrameObject () ;
};

struct I3EventHeader : I3FrameObject {
    int run_id, subrun_id, event_id, etc;
};

typedef I3Frame map<string, I3FrameObjectPtr>;

void MyModule::Physics(I3FramePtr frame)
{
    I3FrameObjectPtr fop = (*frame)['hitseries'];
    if (!fop) throw_exception(...);
    I3EventHeaderPtr eh =
        shared_pointer_cast<I3EventHeader>(fop);
    if (!eh) throw_exception(...);
    ....
```

The I3Frame

```
struct I3FrameObject {
    virtual ~I3FrameObject () ;
};

struct I3EventHeader : I3FrameObject {
    int run_id, subrun_id, event_id, etc;
};

typedef I3Frame map<string, I3FrameObjectPtr>;

void MyModule::Physics(I3FramePtr frame)
{
    I3FrameObjectPtr fop = (*frame)['hitseries'];
    if (!fop) throw_exception(...);
    I3EventHeaderPtr eh =
        shared_pointer_cast<I3EventHeader>(fop);
    if (!eh) throw_exception(...);
    ....
```

The I3Frame

```
struct I3FrameObject {
    virtual ~I3FrameObject () ;
};

struct I3EventHeader : I3FrameObject {
    int run_id, subrun_id, event_id, etc;
};

typedef I3Frame map<string, I3FrameObjectPtr>;

void MyModule::Physics(I3FramePtr frame)
{
    I3FrameObjectPtr fop = (*frame)['hitseries'];
    if (!fop) throw_exception(...);
    I3EventHeaderPtr eh =
        shared_pointer_cast<I3EventHeader>(fop);
    if (!eh) throw_exception(...);
    ...
}
```

The I3Frame

```
struct I3FrameObject {
    virtual ~I3FrameObject () ;
};

struct I3EventHeader : I3FrameObject {
    int run_id, subrun_id, event_id, etc;
};

typedef I3Frame map<string, I3FrameObjectPtr>;

void MyModule::Physics(I3FramePtr frame)
{
    I3FrameObjectPtr fop = (*frame)['hitseries'];
if (!fop) throw_exception(...);
    I3EventHeaderPtr eh =
        shared_pointer_cast<I3EventHeader>(fop);
    if (!eh) throw_exception(...);
    ....
```

The I3Frame

```
struct I3FrameObject {
    virtual ~I3FrameObject () ;
};

struct I3EventHeader : I3FrameObject {
    int run_id, subrun_id, event_id, etc;
};

typedef I3Frame map<string, I3FrameObjectPtr>;

void MyModule::Physics(I3FramePtr frame)
{
    I3FrameObjectPtr fop = (*frame)['hitseries'];
    if (!fop) throw_exception(...);
    I3EventHeaderPtr eh =
        shared_pointer_cast<I3EventHeader>(fop);
    if (!eh) throw_exception(...);
    ....
```

The I3Frame

```
struct I3FrameObject {
    virtual ~I3FrameObject () ;
};

struct I3EventHeader : I3FrameObject {
    int run_id, subrun_id, event_id, etc;
};

typedef I3Frame map<string, I3FrameObjectPtr>;

void MyModule::Physics(I3FramePtr frame)
{
    I3FrameObjectPtr fop = (*frame)['hitseries'];
    if (!fop) throw_exception(...);
    I3EventHeaderPtr eh =
        shared_pointer_cast<I3EventHeader>(fop);
if (!eh) throw_exception(...);
    ....
```

The I3Frame

```
class I3Frame
{
    std::map<string, I3FrameObjectPtr> storage_;

public:

    template <typename T>
    T
    Get(const std::string& name,
        typename boost::enable_if<is_const_shared_ptr<T>
                                ::type* = 0) const;

    template <typename T>
    const T&
    Get(const std::string& name,
        typename boost::disable_if<is_const_shared_ptr<T>
                                ::type* = 0) const;

};

};
```

The I3Frame

```
class I3Frame
{
    std::map<string, I3FrameObjectPtr> storage_;

public:

template <typename T>
T
Get(const std::string& name,
     typename boost::enable_if<is_const_shared_ptr<T>
                           ::type* = 0) const;

template <typename T>
const T&
Get(const std::string& name,
     typename boost::disable_if<is_const_shared_ptr<T>
                           ::type* = 0) const;

};

};
```

The I3Frame

```
class I3Frame
{
    std::map<string, I3FrameObjectPtr> storage_;

public:

    template <typename T>
    T
    Get(const std::string& name,
        typename boost::enable_if<is_const_shared_ptr<T>
                                >::type* = 0) const;

    template <typename T>
    const T&
    Get(const std::string& name,
        typename boost::disable_if<is_const_shared_ptr<T>
                                >::type* = 0) const;

};

};
```

The I3Frame

```
class I3Frame
{
    std::map<string, I3FrameObjectPtr> storage_;

public:

    template <typename T>
    T
    Get(const std::string& name,
        typename boost::enable_if<is_const_shared_ptr<T>
                                ::type* = 0) const;

    template <typename T>
    const T&
    Get(const std::string& name,
        typename boost::disable_if<is_const_shared_ptr<T>
                                ::type* = 0) const;

};

};
```

The I3Frame

```
class I3Frame
{
    std::map<string, I3FrameObjectPtr> storage_;

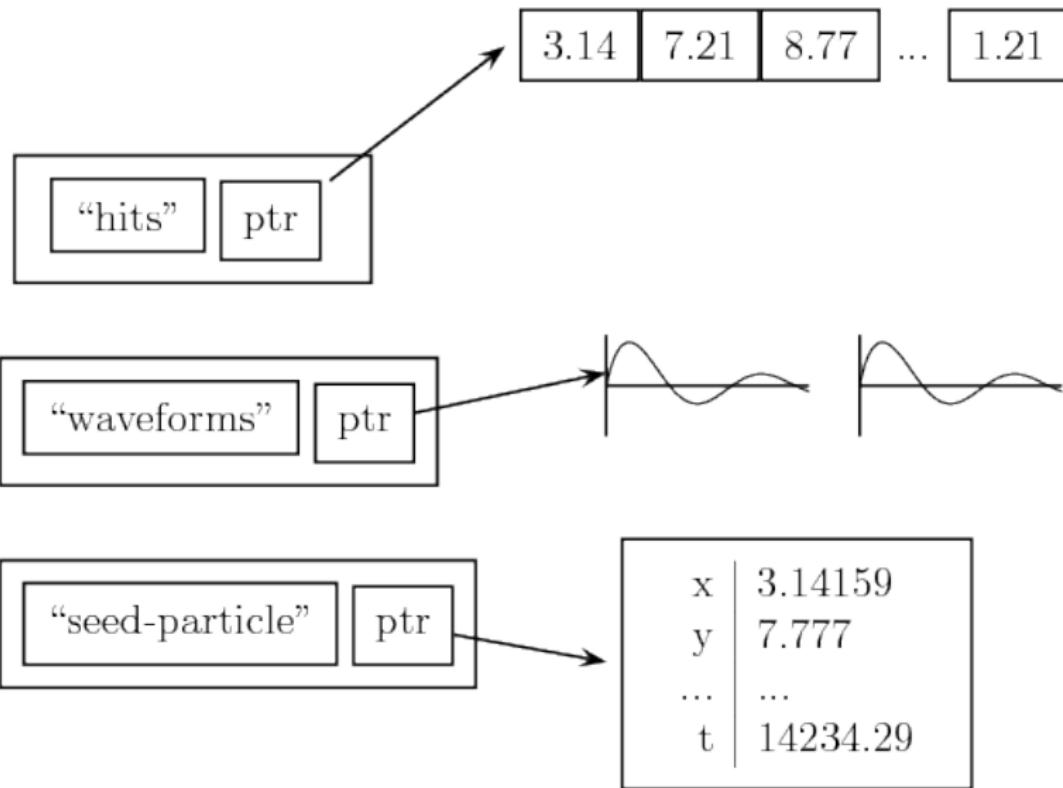
public:
    void
    Put(const std::string& name, I3FrameObjectPtr ptr);

    void
    Has(const std::string& name);

    void
    Delete(const std::string& name);

    void
    Rename(const std::string& from, const std::string& to);
};
```

The I3Frame early on



The I3Frame

```
void MyModule::Physics(I3FramePtr frame)
{
    const I3EventHeader& eh = frame->Get<I3EventHeader>("where")

    I3EventHeaderConstPtr eh_p =
        frame->Get<I3EventHeaderConstPtr>("where");

    I3EventHeaderPtr eh_p(new I3EventHeader);

    eh_p->whatever = 3;

    frame->Put("alt_eventheader", eh_p);
}
```

The I3Frame

```
void MyModule::Physics(I3FramePtr frame)
{
    const I3EventHeader& eh = frame->Get<I3EventHeader>("where")

    I3EventHeaderConstPtr eh_p =
        frame->Get<I3EventHeaderConstPtr>("where");

    I3EventHeaderPtr eh_p(new I3EventHeader);

    eh_p->whatever = 3;

    frame->Put("alt_eventheader", eh_p);
}
```

The I3Frame

```
void MyModule::Physics(I3FramePtr frame)
{
    const I3EventHeader& eh = frame->Get<I3EventHeader>("where")

    I3EventHeaderConstPtr eh_p =
        frame->Get<I3EventHeaderConstPtr>("where");

I3EventHeaderPtr eh_p(new I3EventHeader);

    eh_p->whatever = 3;

    frame->Put("alt_eventheader", eh_p);
}
```

The I3Frame

```
void MyModule::Physics(I3FramePtr frame)
{
    const I3EventHeader& eh = frame->Get<I3EventHeader>("where")

    I3EventHeaderConstPtr eh_p =
        frame->Get<I3EventHeaderConstPtr>("where");

    I3EventHeaderPtr eh_p(new I3EventHeader);

eh_p->whatever = 3;

    frame->Put("alt_eventheader", eh_p);
}
```

The I3Frame

```
void MyModule::Physics(I3FramePtr frame)
{
    const I3EventHeader& eh = frame->Get<I3EventHeader>("where")

    I3EventHeaderConstPtr eh_p =
        frame->Get<I3EventHeaderConstPtr>("where");

    I3EventHeaderPtr eh_p(new I3EventHeader);

    eh_p->whatever = 3;

    frame->Put("alt_eventheader", eh_p);
}
```

Recap

- ▶ There is a C++ framework that loads shared libraries containing I3Modules
- ▶ These modules Put/Get I3FrameObjects to/from an I3Frame
- ▶ Shared libraries may define classes that inherit from I3FrameObject and put them into passing frames.
- ▶ Everything is held by `boost::shared_ptr<>`

Recap

- ▶ There is a C++ framework that loads shared libraries containing I3Modules
- ▶ These modules Put/Get I3FrameObjects to/from an I3Frame
- ▶ Shared libraries may define classes that inherit from I3FrameObject and put them into passing frames.
- ▶ Everything is held by `boost::shared_ptr<>`

Recap

- ▶ There is a C++ framework that loads shared libraries containing I3Modules
- ▶ These modules Put/Get I3FrameObjects to/from an I3Frame
- ▶ Shared libraries may define classes that inherit from I3FrameObject and put them into passing frames.
- ▶ Everything is held by `boost::shared_ptr<>`

Recap

- ▶ There is a C++ framework that loads shared libraries containing I3Modules
- ▶ These modules Put/Get I3FrameObjects to/from an I3Frame
- ▶ Shared libraries may define classes that inherit from I3FrameObject and put them into passing frames.
- ▶ Everything is held by `boost::shared_ptr<>`

boost::serialization

A serializable struct

```
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year

    template <typename Archive>
    void
    serialize(Archive& ar, unsigned version)
    {
        ar & make_nvp("year", year);
        ar & make_nvp("daq_time", daq_time);
    }
};
```

A serializable struct

```
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                          // since start of year

    template <typename Archive>
    void
    serialize(Archive& ar, unsigned version)
    {
        ar & make_nvp("year", year);
        ar & make_nvp("daq_time", daq_time);
    }
};
```

A serializable struct

```
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year

    template <typename Archive>
    void
    serialize(Archive& ar, unsigned version)
    {
        ar & make_nvp("year", year);
        ar & make_nvp("daq_time", daq_time);
    }
};
```

A serializable struct

```
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year

    template <typename Archive>
void
    serialize(Archive& ar, unsigned version)
    {
        ar & make_nvp("year", year);
        ar & make_nvp("daq_time", daq_time);
    }
};
```

A serializable struct

```
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year

    template <typename Archive>
    void
    serialize(Archive& ar, unsigned version)
    {
        ar & make_nvp("year", year);
        ar & make_nvp("daq_time", daq_time);
    }
};
```

A serializable struct

```
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year

    template <typename Archive>
    void
    serialize(Archive& ar, unsigned version)
    {
        ar & make_nvp("year", year);
        ar & make_nvp("daq_time", daq_time);
    }
};
```

A serializable struct

```
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year

    template <typename Archive>
    void
    serialize(Archive& ar, unsigned version)
    {
        ar & make_nvp("year", year);
        ar & make_nvp("daq_time", daq_time);
    }
};
```

A serializable struct

```
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year

    template <typename Archive>
    void
    serialize(Archive& ar, unsigned version)
    {
        ar & make_nvp("year", year);
        ar & make_nvp("daq_time", daq_time);
    }
};
```

A serializable struct

```
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year

    template <typename Archive>
    void
    serialize(Archive& ar, unsigned version)
    {
        ar & make_nvp("year", year);
        ar & make_nvp("daq_time", daq_time);
    }
};
```

Serializing this struct

```
I3Time t;  
{  
    xml_oarchive xoa(std::cout);  
    xoa << make_nvp("the_time", t);  
}
```

Serializing this struct

```
I3Time t;  
{  
    xml_oarchive xoa(std::cout);  
    xoa << make_nvp("the_time", t);  
}
```

Serializing this struct

```
I3Time t;  
{  
    xml_oarchive xoa(std::cout);  
    xoa << make_nvp("the_time", t);  
}
```

Serializing this struct

```
I3Time t;  
{  
    xml_oarchive xoa(std::cout);  
    xoa << make_nvp("the_time", t);  
}
```

Serializing this struct

```
I3Time t;  
{  
    xml_oarchive xoa(std::cout);  
    xoa << make_nvp("the_time", t);  
}
```

Serializing this struct

```
I3Time t;  
{  
    xml_oarchive xoa(std::cout);  
    xoa & make_nvp("the_time", t);  
}
```

Serializing this struct

```
I3Time t;  
{  
    xml_oarchive xoa(std::cout);  
    xoa & make_nvp("the_time", t);  
}
```

```
<the_time>  
  <year>2007</year>  
  <daq_time>131398140230407991</daq_time>  
</the_time>
```

And they nest!

```
struct TimeRange
{
    I3Time start, end;

    template <typename Archive>
    void
    serialize(Archive& ar, unsigned version)
    {
        ar & make_nvp("start", start);
        ar & make_nvp("end", end);
    }
};

TimeRange tr;
my_archive << make_nvp("range", tr);
```

And they nest!

```
struct TimeRange
{
    I3Time start, end;

    template <typename Archive>
    void
    serialize(Archive& ar, unsigned version)
    {
        ar & make_nvp("start", start);
        ar & make_nvp("end", end);
    }
};

TimeRange tr;
my_archive << make_nvp("range", tr);
```

And they nest!

```
struct TimeRange
{
    I3Time start, end;

    template <typename Archive>
    void
    serialize(Archive& ar, unsigned version)
    {
        ar & make_nvp("start", start);
        ar & make_nvp("end", end);
    }
};

TimeRange tr;
my_archive << make_nvp("range", tr);
```

And they nest!

```
struct TimeRange
{
    I3Time start, end;

    template <typename Archive>
    void
    serialize(Archive& ar, unsigned version)
    {
        ar & make_nvp("start", start);
        ar & make_nvp("end", end);
    }
};

TimeRange tr;
my_archive << make_nvp("range", tr);
```

And they nest!

```
struct TimeRange
{
    I3Time start, end;

    template <typename Archive>
    void
    serialize(Archive& ar, unsigned version)
    {
        ar & make_nvp("start", start);
        ar & make_nvp("end", end);
    }
};

TimeRange tr;
my_archive << make_nvp("range", tr);
```

And they nest!

```
<boost_serialization signature="serialization::archive"
                     version="5">
  <range>
    <start>
      <year>2007</year>
      <daq_time>131398140230407991</daq_time>
    </start>
    <end>
      <year>2008</year>
      <daq_time>731366640230401997</daq_time>
    </end>
  </range>
</boost_serialization>
```

And they nest!

```
<boost_serialization signature="serialization::archive"
                     version="5">
  <range>
    <start>
      <year>2007</year>
      <daq_time>131398140230407991</daq_time>
    </start>
    <end>
      <year>2008</year>
      <daq_time>731366640230401997</daq_time>
    </end>
  </range>
</boost_serialization>
```

The same thing... nonintrusively

```
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year

    template <typename Archive>
    void
    serialize(Archive& ar, unsigned version)
    {
        ar & make_nvp("year", year);
        ar & make_nvp("daq_time", daq_time);
    }
};
```

The same thing... nonintrusively

```
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year

    template <typename Archive>
    void
    serialize(Archive& ar, unsigned version)
    {
        ar & make_nvp("year", year);
        ar & make_nvp("daq_time", daq_time);
    }
};
```

The same thing... nonintrusively

```
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year
};

template <typename Archive>
void
serialize(Archive& ar, I3Time& time, unsigned version)
{
    ar & make_nvp("year", time.year);
    ar & make_nvp("daq_time", time.daq_time);
}
```

Practicing good header hygiene

```
#include <boost/serialization/nvp.hpp>
// and potentially many others
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year

    template <typename Archive>
    void
    serialize(Archive& ar, unsigned version)
    {
        ar & make_nvp("year", year);
        ar & make_nvp("daq_time", daq_time);
    }
};
```

Practicing good header hygiene

```
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year

    template <typename Archive>
    void
    serialize(Archive& ar, unsigned version);

};
```

Practicing good header hygiene: serialization.hpp

```
#include <boost/serialization/nvp.hpp>
#include <boost/serialization/base_object.hpp>
#include <boost/archive/xml_oarchive.hpp>
#include <boost/archive/xml_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
// etc

#define I3_SERIALIZABLE(T) \
template void T::serialize(boost::archive::binary_oarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::binary_iarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::xml_iarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::xml_oarchive&, \
                           unsigned); \
BOOST_CLASS_EXPORT(T);
```

Practicing good header hygiene: serialization.hpp

```
#include <boost/serialization/nvp.hpp>
#include <boost/serialization/base_object.hpp>
#include <boost/archive/xml_oarchive.hpp>
#include <boost/archive/xml_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
// etc

#define I3_SERIALIZABLE(T) \
template void T::serialize(boost::archive::binary_oarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::binary_iarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::xml_iarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::xml_oarchive&, \
                           unsigned); \
BOOST_CLASS_EXPORT(T);
```

Practicing good header hygiene: serialization.hpp

```
#include <boost/serialization/nvp.hpp>
#include <boost/serialization/base_object.hpp>
#include <boost/archive/xml_oarchive.hpp>
#include <boost/archive/xml_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
// etc

#define I3_SERIALIZABLE(T) \
template void T::serialize(boost::archive::binary_oarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::binary_iarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::xml_iarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::xml_oarchive&, \
                           unsigned); \
BOOST_CLASS_EXPORT(T);
```

Practicing good header hygiene: serialization.hpp

```
#include <boost/serialization/nvp.hpp>
#include <boost/serialization/base_object.hpp>
#include <boost/archive/xml_oarchive.hpp>
#include <boost/archive/xml_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
// etc

#define I3_SERIALIZABLE(T) \
template void T::serialize(boost::archive::binary_oarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::binary_iarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::xml_iarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::xml_oarchive&, \
                           unsigned); \
BOOST_CLASS_EXPORT(T);
```

Practicing good header hygiene: serialization.hpp

```
#include <boost/serialization/nvp.hpp>
#include <boost/serialization/base_object.hpp>
#include <boost/archive/xml_oarchive.hpp>
#include <boost/archive/xml_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
// etc

#define I3_SERIALIZABLE(T) \
template void T::serialize(boost::archive::binary_oarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::binary_iarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::xml_iarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::xml_oarchive&, \
                           unsigned); \
BOOST_CLASS_EXPORT(T);
```

Practicing good header hygiene: serialization.hpp

```
#include <boost/serialization/nvp.hpp>
#include <boost/serialization/base_object.hpp>
#include <boost/archive/xml_oarchive.hpp>
#include <boost/archive/xml_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
// etc

#define I3_SERIALIZABLE(T) \
template void T::serialize(boost::archive::binary_oarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::binary_iarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::xml_iarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::xml_oarchive&, \
                           unsigned); \
BOOST_CLASS_EXPORT(T);
```

Practicing good header hygiene: serialization.hpp

```
#include <boost/serialization/nvp.hpp>
#include <boost/serialization/base_object.hpp>
#include <boost/archive/xml_oarchive.hpp>
#include <boost/archive/xml_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
// etc

#define I3_SERIALIZABLE(T) \
template void T::serialize(boost::archive::binary_oarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::binary_iarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::xml_iarchive&, \
                           unsigned); \
template void T::serialize(boost::archive::xml_oarchive&, \
                           unsigned); \
BOOST_CLASS_EXPORT(T);
```

Good hygiene continued

```
#include <icetray/serialization.hpp>    // the utility header

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    ar & make_nvp("year", year);
    ar & make_nvp("daq_time", daq_time);
}

I3_SERIALIZABLE(I3Time);
```

Good hygiene continued

```
#include <icetray/serialization.hpp>    // the utility header

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    ar & make_nvp("year", year);
    ar & make_nvp("daq_time", daq_time);
}

I3_SERIALIZABLE(I3Time);
```

Good hygiene continued

```
#include <icetray/serialization.hpp>      // the utility header

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    ar & make_nvp("year", year);
    ar & make_nvp("daq_time", daq_time);
}

I3_SERIALIZABLE(I3Time);
```

Good hygiene continued

```
#include <icetray/serialization.hpp>    // the utility header

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    ar & make_nvp("year", year);
    ar & make_nvp("daq_time", daq_time);
}

I3_SERIALIZABLE(I3Time);
```

Canned routines: std::vector

```
#include <boost/serialization/vector.hpp>
#include <boost/archive/xml_oarchive.hpp>

std::vector<int> v(3, 7);
{
    boost::archive::xml_oarchive xoa(std::cout);
    xoa << make_nvp("v", v);
}

<!DOCTYPE boost_serialization>
<boost_serialization signature="serialization::archive"
                      version="5">
<v>
    <count>3</count>
    <item>7</item>
    <item>7</item>
    <item>7</item>
</v>
</boost_serialization>
```

Canned routines: std::vector

```
#include <boost/serialization/vector.hpp>
#include <boost/archive/xml_oarchive.hpp>

std::vector<int> v(3, 7);
{
    boost::archive::xml_oarchive xoa(std::cout);
    xoa << make_nvp("v", v);
}

<!DOCTYPE boost_serialization>
<boost_serialization signature="serialization::archive"
                      version="5">
<v>
    <count>3</count>
    <item>7</item>
    <item>7</item>
    <item>7</item>
</v>
</boost_serialization>
```

Canned routines: std::vector

```
#include <boost/serialization/vector.hpp>
#include <boost/archive/xml_oarchive.hpp>

std::vector<int> v(3, 7);
{
    boost::archive::xml_oarchive xoa(std::cout);
    xoa << make_nvp("v", v);
}

<!DOCTYPE boost_serialization>
<boost_serialization signature="serialization::archive"
                      version="5">
<v>
    <count>3</count>
    <item>7</item>
    <item>7</item>
    <item>7</item>
</v>
</boost_serialization>
```

Canned routines: std::vector

```
#include <boost/serialization/vector.hpp>
#include <boost/archive/xml_oarchive.hpp>

std::vector<int> v(3, 7);
{
    boost::archive::xml_oarchive xoa(std::cout);
    xoa << make_nvp("v", v);
}

<!DOCTYPE boost_serialization>
<boost_serialization signature="serialization::archive"
                      version="5">
<v>
    <count>3</count>
    <item>7</item>
    <item>7</item>
    <item>7</item>
</v>
</boost_serialization>
```

Canned routines: std::vector

```
#include <boost/serialization/vector.hpp>
#include <boost/archive/xml_oarchive.hpp>

std::vector<int> v(3, 7);
{
    boost::archive::xml_oarchive xoa(std::cout);
    xoa << make_nvp("v", v);
}

<!DOCTYPE boost_serialization>
<boost_serialization signature="serialization::archive"
                      version="5">
<v>
    <count>3</count>
    <item>7</item>
    <item>7</item>
    <item>7</item>
</v>
</boost_serialization>
```

Canned routines: std::map

```
#include <boost/serialization/map.hpp>
#include <boost/archive/xml_oarchive.hpp>

std::map<std::string, int> m; m["one"] = 1; m["two"] = 2;
boost::archive::xml_oarchive xoa(std::cout);
xoa << make_nvp("m", m);

<m class_id="0" tracking_level="0" version="0">
    <count>2</count>
    <item class_id="1" tracking_level="0" version="0">
        <first>one</first>
        <second>1</second>
    </item>
    <item>
        <first>two</first>
        <second>2</second>
    </item>
</m>
```

Canned routines: std::map

```
#include <boost/serialization/map.hpp>
#include <boost/archive/xml_oarchive.hpp>

std::map<std::string, int> m; m["one"] = 1; m["two"] = 2;
boost::archive::xml_oarchive xoa(std::cout);
xoa << make_nvp("m", m);

<m class_id="0" tracking_level="0" version="0">
    <count>2</count>
    <item class_id="1" tracking_level="0" version="0">
        <first>one</first>
        <second>1</second>
    </item>
    <item>
        <first>two</first>
        <second>2</second>
    </item>
</m>
```

Canned routines: std::map

```
#include <boost/serialization/map.hpp>
#include <boost/archive/xml_oarchive.hpp>

std::map<std::string, int> m; m["one"] = 1; m["two"] = 2;
boost::archive::xml_oarchive xoa(std::cout);
xoa << make_nvp("m", m);

<m class_id="0" tracking_level="0" version="0">
    <count>2</count>
    <item class_id="1" tracking_level="0" version="0">
        <first>one</first>
        <second>1</second>
    </item>
    <item>
        <first>two</first>
        <second>2</second>
    </item>
</m>
```

Canned routines: std::map

```
#include <boost/serialization/map.hpp>
#include <boost/archive/xml_oarchive.hpp>

std::map<std::string, int> m; m["one"] = 1; m["two"] = 2;
boost::archive::xml_oarchive xoa(std::cout);
xoa << make_nvp("m", m);

<m class_id="0" tracking_level="0" version="0">
    <count>2</count>
    <item class_id="1" tracking_level="0" version="0">
        <first>one</first>
        <second>1</second>
    </item>
    <item>
        <first>two</first>
        <second>2</second>
    </item>
</m>
```

Canned routines: std::map

```
#include <boost/serialization/map.hpp>
#include <boost/archive/xml_oarchive.hpp>

std::map<std::string, int> m; m["one"] = 1; m["two"] = 2;
boost::archive::xml_oarchive xoa(std::cout);
xoa << make_nvp("m", m);

<m class_id="0" tracking_level="0" version="0">
    <count>2</count>
    <item class_id="1" tracking_level="0" version="0">
        <first>one</first>
        <second>1</second>
    </item>
    <item>
        <first>two</first>
        <second>2</second>
    </item>
</m>
```

boost::variant

```
#include <boost/serialization/variant.hpp>
#include <boost/archive/xml_oarchive.hpp>

boost::archive::xml_oarchive xoa(std::cout);

boost::variant<int, double, std::string> v;
v = 3.14159;

xoa << make_nvp("v", v);

<v class_id="0" tracking_level="0" version="0">
    <which>1</which>
    <value>3.1415899999999999</value>
</v>
```

boost::variant

```
#include <boost/serialization/variant.hpp>
#include <boost/archive/xml_oarchive.hpp>

boost::archive::xml_oarchive xoa(std::cout);

boost::variant<int, double, std::string> v;
v = 3.14159;

xoa << make_nvp("v", v);

<v class_id="0" tracking_level="0" version="0">
    <which>1</which>
    <value>3.1415899999999999</value>
</v>
```

boost::variant

```
#include <boost/serialization/variant.hpp>
#include <boost/archive/xml_oarchive.hpp>

boost::archive::xml_oarchive xoa(std::cout);

boost::variant<int, double, std::string> v;
v = 3.14159;

xoa << make_nvp("v", v);

<v class_id="0" tracking_level="0" version="0">
    <which>1</which>
    <value>3.1415899999999999</value>
</v>
```

boost::variant

```
#include <boost/serialization/variant.hpp>
#include <boost/archive/xml_oarchive.hpp>

boost::archive::xml_oarchive xoa(std::cout);

boost::variant<int, double, std::string> v;
v = 3.14159;

xoa << make_nvp("v", v);

<v class_id="0" tracking_level="0" version="0">
    <which>1</which>
    <value>3.141589999999999</value>
</v>
```

boost::variant

```
#include <boost/serialization/variant.hpp>
#include <boost/archive/xml_oarchive.hpp>

boost::archive::xml_oarchive xoa(std::cout);

boost::variant<int, double, std::string> v;
v = 3.14159;

xoa << make_nvp("v", v);

<v class_id="0" tracking_level="0" version="0">
    <which>1</which>
    <value>3.141589999999999</value>
</v>
```

boost::shared_ptr to variant

```
typedef boost::variant<int, double, std::string> var_t;

shared_ptr<var_t> p1(new var_t);
*p1 = 3.14159;
shared_ptr<var_t> p2 = p1;

boost::archive::xml_oarchive xoa(std::cout);
xoa <> make_nvp("p1", p1) <> make_nvp("p2", p2);

<p1 class_id="0" tracking_level="0" version="1">
  <px class_id="1" tracking_level="1" version="0"
      object_id="_0">
    <which>1</which>
    <value>3.141589999999999</value>
  </px>
</p1>
<p2>
  <px class_id_reference="1" object_id_reference="_0"></px>
</p2>
```

boost::shared_ptr to variant

```
typedef boost::variant<int, double, std::string> var_t;

shared_ptr<var_t> p1(new var_t);
*p1 = 3.14159;
shared_ptr<var_t> p2 = p1;

boost::archive::xml_oarchive xoa(std::cout);
xoa <> make_nvp("p1", p1) <> make_nvp("p2", p2);

<p1 class_id="0" tracking_level="0" version="1">
  <px class_id="1" tracking_level="1" version="0"
      object_id="_0">
    <which>1</which>
    <value>3.141589999999999</value>
  </px>
</p1>
<p2>
  <px class_id_reference="1" object_id_reference="_0"></px>
</p2>
```

boost::shared_ptr to variant

```
typedef boost::variant<int, double, std::string> var_t;

shared_ptr<var_t> p1(new var_t);
*p1 = 3.14159;
shared_ptr<var_t> p2 = p1;

boost::archive::xml_oarchive xoa(std::cout);
xoa <> make_nvp("p1", p1) <> make_nvp("p2", p2);

<p1 class_id="0" tracking_level="0" version="1">
  <px class_id="1" tracking_level="1" version="0"
      object_id="_0">
    <which>1</which>
    <value>3.141589999999999</value>
  </px>
</p1>
<p2>
  <px class_id_reference="1" object_id_reference="_0"></px>
</p2>
```

boost::shared_ptr to variant

```
typedef boost::variant<int, double, std::string> var_t;

shared_ptr<var_t> p1(new var_t);
*p1 = 3.14159;
shared_ptr<var_t> p2 = p1;

boost::archive::xml_oarchive xoa(std::cout);
xoa <> make_nvp("p1", p1) <> make_nvp("p2", p2);

<p1 class_id="0" tracking_level="0" version="1">
  <px class_id="1" tracking_level="1" version="0"
      object_id="_0">
    <which>1</which>
    <value>3.141589999999999</value>
  </px>
</p1>
<p2>
  <px class_id_reference="1" object_id_reference="_0"></px>
</p2>
```

boost::shared_ptr to variant

```
typedef boost::variant<int, double, std::string> var_t;

shared_ptr<var_t> p1(new var_t);
*p1 = 3.14159;
shared_ptr<var_t> p2 = p1;

boost::archive::xml_oarchive xoa(std::cout);
xoa <> make_nvp("p1", p1) <> make_nvp("p2", p2);

<p1 class_id="0" tracking_level="0" version="1">
  <px class_id="1" tracking_level="1" version="0"
      object_id="_0">
    <which>1</which>
    <value>3.141589999999999</value>
  </px>
</p1>
<p2>
  <px class_id_reference="1" object_id_reference="_0"></px>
</p2>
```

boost::shared_ptr to variant

```
typedef boost::variant<int, double, std::string> var_t;

shared_ptr<var_t> p1(new var_t);
*p1 = 3.14159;
shared_ptr<var_t> p2 = p1;

boost::archive::xml_oarchive xoa(std::cout);
xoa <> make_nvp("p1", p1) <> make_nvp("p2", p2);

<p1 class_id="0" tracking_level="0" version="1">
  <px class_id="1" tracking_level="1" version="0"
      object_id="_0">
    <which>1</which>
    <value>3.141589999999999</value>
  </px>
</p1>
<p2>
  <px class_id_reference="1" object_id_reference="_0"></px>
</p2>
```

boost::shared_ptr to variant

```
typedef boost::variant<int, double, std::string> var_t;

shared_ptr<var_t> p1(new var_t);
*p1 = 3.14159;
shared_ptr<var_t> p2 = p1;

boost::archive::xml_oarchive xoa(std::cout);
xoa <> make_nvp("p1", p1) <> make_nvp("p2", p2);

<p1 class_id="0" tracking_level="0" version="1">
  <px class_id="1" tracking_level="1" version="0"
      object_id="_0">
    <which>1</which>
    <value>3.141589999999999</value>
  </px>
</p1>
<p2>
  <px class_id_reference="1" object_id_reference="_0"></px>
</p2>
```

boost::shared_ptr to variant

```
typedef boost::variant<int, double, std::string> var_t;

shared_ptr<var_t> p1(new var_t);
*p1 = 3.14159;
shared_ptr<var_t> p2 = p1;

boost::archive::xml_oarchive xoa(std::cout);
xoa <> make_nvp("p1", p1) <> make_nvp("p2", p2);

<p1 class_id="0" tracking_level="0" version="1">
<px class_id="1" tracking_level="1" version="0"
    object_id="_0">
    <which>1</which>
    <value>3.141589999999999</value>
</px>
</p1>
<p2>
    <px class_id_reference="1" object_id_reference="_0"></px>
</p2>
```

boost::shared_ptr to variant

```
typedef boost::variant<int, double, std::string> var_t;

shared_ptr<var_t> p1(new var_t);
*p1 = 3.14159;
shared_ptr<var_t> p2 = p1;

boost::archive::xml_oarchive xoa(std::cout);
xoa <> make_nvp("p1", p1) <> make_nvp("p2", p2);

<p1 class_id="0" tracking_level="0" version="1">
  <px class_id="1" tracking_level="1" version="0"
      object_id="_0"
```

boost::shared_ptr to variant

```
typedef boost::variant<int, double, std::string> var_t;

shared_ptr<var_t> p1(new var_t);
*p1 = 3.14159;
shared_ptr<var_t> p2 = p1;

boost::archive::xml_oarchive xoa(std::cout);
xoa <> make_nvp("p1", p1) <> make_nvp("p2", p2);

<p1 class_id="0" tracking_level="0" version="1">
<px class_id="1" tracking_level="1" version="0"
     object_id="_0">
  <which>1</which>
  <value>3.141589999999999</value>
</px>
</p1>
<p2>
  <px class_id_reference="1" object_id_reference="_0"></px>
</p2>
```

boost::shared_ptr to variant

```
typedef boost::variant<int, double, std::string> var_t;

shared_ptr<var_t> p1(new var_t);
*p1 = 3.14159;
shared_ptr<var_t> p2 = p1;

boost::archive::xml_oarchive xoa(std::cout);
xoa <> make_nvp("p1", p1) <> make_nvp("p2", p2);

<p1 class_id="0" tracking_level="0" version="1">
  <px class_id="1" tracking_level="1" version="0"
      object_id="_0">
    <which>1</which>
    <value>3.141589999999999</value>
  </px>
</p1>
<p2>
  <px class_id_reference="1" object_id_reference="_0"></px>
</p2>
```

boost::shared_ptr to variant

```
typedef boost::variant<int, double, std::string> var_t;

shared_ptr<var_t> p1(new var_t);
*p1 = 3.14159;
shared_ptr<var_t> p2 = p1;

boost::archive::xml_oarchive xoa(std::cout);
xoa <> make_nvp("p1", p1) <> make_nvp("p2", p2);

<p1 class_id="0" tracking_level="0" version="1">
  <px class_id="1" tracking_level="1" version="0"
      object_id="_0">
    <which>1</which>
    <value>3.141589999999999</value>
  </px>
</p1>
<p2>
  <px class_id_reference="1" object_id_reference="_0"></px>
</p2>
```

boost::shared_ptr to variant

```
typedef boost::variant<int, double, std::string> var_t;

shared_ptr<var_t> p1(new var_t);
*p1 = 3.14159;
shared_ptr<var_t> p2 = p1;

boost::archive::xml_oarchive xoa(std::cout);
xoa <> make_nvp("p1", p1) <> make_nvp("p2", p2);

<p1 class_id="0" tracking_level="0" version="1">
  <px class_id="1" tracking_level="1" version="0"
      object_id="_0">
    <which>1</which>
    <value>3.141589999999999</value>
  </px>
</p1>
<p2>
  <px class_id_reference="1" object_id_reference="_0"></px>
</p2>
```

Serialization via pointer-to-base

```
I3FrameObjectPtr obj_p(new I3Time);

{
    xml_oarchive xoa(std::cout);
    xoa & make_nvp("obj_p", obj_p);
}
```

Serialization via pointer-to-base

```
I3FrameObjectPtr obj_p(new I3Time);  
  
{  
    xml_oarchive xoa(std::cout);  
    xoa & make_nvp("obj_p", obj_p);  
}
```

Serialization via pointer-to-base

```
I3FrameObjectPtr obj_p(new I3Time);

{
    xml_oarchive xoa(std::cout);
    xoa & make_nvp("obj_p", obj_p);
}
```

Serialization via pointer-to-base

```
I3FrameObjectPtr obj_p(new I3Time);

{
    xml_oarchive xoa(std::cout);
    xoa & make_nvp("obj_p", obj_p);
}
```

I3FrameObject.hpp

```
struct I3FrameObject
{
    virtual ~I3FrameObject();

    template <class Archive>
    void
    serialize(Archive& ar, unsigned version);

};
```

I3FrameObject.hpp

```
struct I3FrameObject
{
    virtual ~I3FrameObject();

    template <class Archive>
    void
    serialize(Archive& ar, unsigned version);

};
```

I3FrameObject.hpp

```
struct I3FrameObject
{
    virtual ~I3FrameObject();

template <class Archive>
void
serialize(Archive& ar, unsigned version);

};
```

I3FrameObject.cpp

```
I3FrameObject::~I3FrameObject() { }

template <typename Archive>
void
I3FrameObject::serialize(Archive& ar, unsigned version)
{ }

I3_SERIALIZABLE(I3FrameObject);
BOOST_SERIALIZATION_SHARED_PTR(I3FrameObject);
```

I3FrameObject.cpp

```
I3FrameObject::~I3FrameObject() { }

template <typename Archive>
void
I3FrameObject::serialize(Archive& ar, unsigned version)
{ }

I3_SERIALIZABLE(I3FrameObject);
BOOST_SERIALIZATION_SHARED_PTR(I3FrameObject);
```

I3FrameObject.cpp

```
I3FrameObject::~I3FrameObject() { }

template <typename Archive>
void
I3FrameObject::serialize(Archive& ar, unsigned version)
{ }

I3_SERIALIZABLE(I3FrameObject);
BOOST_SERIALIZATION_SHARED_PTR(I3FrameObject);
```

I3FrameObject.cpp

```
I3FrameObject::~I3FrameObject() { }

template <typename Archive>
void
I3FrameObject::serialize(Archive& ar, unsigned version)
{ }

I3_SERIALIZABLE(I3FrameObject);
BOOST_SERIALIZATION_SHARED_PTR(I3FrameObject);
```

I3FrameObject.cpp

```
I3FrameObject::~I3FrameObject() { }

template <typename Archive>
void
I3FrameObject::serialize(Archive& ar, unsigned version)
{ }

I3_SERIALIZABLE(I3FrameObject);
BOOST_SERIALIZATION_SHARED_PTR(I3FrameObject);
```

I3Time with base class

```
struct I3Time : public I3FrameObject
{
    ...

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    ar & make_nvp("base",
                  base_object<I3FrameObject>(*this));
    ar & make_nvp("year", year);
    ar & make_nvp("daq_time", daq_time);
}

I3_SERIALIZABLE(I3Time);
```

I3Time with base class

```
struct I3Time : public I3FrameObject
{
    ...

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    ar & make_nvp("base",
                  base_object<I3FrameObject>(*this));
    ar & make_nvp("year", year);
    ar & make_nvp("daq_time", daq_time);
}

I3_SERIALIZABLE(I3Time);
```

I3Time with base class

```
struct I3Time : public I3FrameObject
{
    ...

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    ar & make_nvp("base",
                  base_object<I3FrameObject>(*this));
    ar & make_nvp("year", year);
    ar & make_nvp("daq_time", daq_time);
}

I3_SERIALIZABLE(I3Time);
```

Serialization via pointer-to-base

```
I3FrameObjectPtr obj_p(new I3Time);  
  
{  
    xml_oarchive xoa(std::cout);  
    xoa & make_nvp("I3FrameObject", obj_p);  
}
```

Serialization via pointer-to-base

```
I3FrameObjectPtr obj_p(new I3Time);  
  
{  
    xml_oarchive xoa(std::cout);  
    xoa & make_nvp("I3FrameObject", obj_p);  
}
```

```
<I3FrameObject class_name="I3Time" version="0">  
    <year>2010</year>  
    <daq_time>131398140000000000</daq_time>  
</I3FrameObject>
```

Serialization via pointer-to-base

```
I3FrameObjectPtr obj_p(new I3Time);  
  
{  
    xml_oarchive xoa(std::cout);  
    xoa & make_nvp("I3FrameObject", obj_p);  
}
```

```
<I3FrameObject class_name="I3Time" version="0">  
    <year>2010</year>  
    <daq_time>131398140000000000</daq_time>  
</I3FrameObject>
```

Serialization via pointer-to-base

```
I3FrameObjectPtr obj_p(new I3Time);  
  
{  
    xml_oarchive xoa(std::cout);  
    xoa & make_nvp("I3FrameObject", obj_p);  
}  
  
BOOST_CLASS_EXPORT(I3Time)  
  
<I3FrameObject class_name="I3Time" version="0">  
    <year>2010</year>  
    <daq_time>131398140000000000</daq_time>  
</I3FrameObject>
```

Serialization via pointer-to-base

```
I3FrameObjectPtr obj_p(new I3Time);  
  
{  
    xml_oarchive xoa(std::cout);  
    xoa & make_nvp("I3FrameObject", obj_p);  
}
```

```
BOOST_CLASS_EXPORT(I3Time)
```

```
<I3FrameObject class_name="I3Time" version="0">  
    <year>2010</year>  
    <daq_time>131398140000000000</daq_time>  
</I3FrameObject>
```

Serialization via pointer-to-base

```
I3FrameObjectPtr obj_p(new I3Time);  
  
{  
    xml_oarchive xoa(std::cout);  
    xoa & make_nvp("I3FrameObject", obj_p);  
}  
  
typedef I3Time IcecubeTime;  
BOOST_CLASS_EXPORT(IcecubeTime)  
  
<I3FrameObject class_name="I3Time" version="0">  
    <year>2010</year>  
    <daq_time>131398140000000000</daq_time>  
</I3FrameObject>
```

Serialization via pointer-to-base

```
I3FrameObjectPtr obj_p(new I3Time);  
  
{  
    xml_oarchive xoa(std::cout);  
    xoa & make_nvp("I3FrameObject", obj_p);  
}  
  
typedef I3Time IcecubeTime; // boom.  
BOOST_CLASS_EXPORT(IcecubeTime)  
  
<I3FrameObject class_name="I3Time" version="0">  
    <year>2010</year>  
    <daq_time>131398140000000000</daq_time>  
</I3FrameObject>
```

Non-default constructor

```
class I3Int : public I3FrameObject {
public:
    int value;

    I3Int(int v) : value(v) { }

    template <typename Archive>
    void
    serialize(Archive& ar, const unsigned version);

private:
    I3Int();
};
```

Non-default constructor

```
class I3Int : public I3FrameObject {  
public:  
    int value;  
  
    I3Int(int v) : value(v) {}  
  
    template <typename Archive>  
    void  
    serialize(Archive& ar, const unsigned version);  
  
private:  
    I3Int();  
};
```

Non-default constructor

```
class I3Int : public I3FrameObject {  
public:  
    int value;  
  
    I3Int(int v) : value(v) {}  
  
    template <typename Archive>  
    void  
    serialize(Archive& ar, const unsigned version);  
  
private:  
    I3Int();  
};
```

Non-default constructor: impl

```
template <typename Archive>
void
I3Int::serialize(Archive &ar, const unsigned version)
{
    ar & make_nvp("base", base_object<I3FrameObject>(*this));
}
```

Non-default constructor: impl

```
namespace boost { namespace serialization {
    template <typename Archive>
    inline void
    save_construct_data(Archive& ar, const I3Int* i3i,
                        unsigned)
    {
        ar << make_nvp("value", i3i->value);
    }

    template <typename Archive>
    inline void
    load_construct_data(Archive& ar, I3Int* i3i,
                        unsigned)
    {
        int value;
        ar >> make_nvp("value", value);
        ::new(i3i) I3Int(value);
    }
}}
```

Non-default constructor: impl

```
namespace boost { namespace serialization {
    template <typename Archive>
    inline void
    save_construct_data(Archive& ar, const I3Int* i3i,
                        unsigned)
    {
        ar << make_nvp("value", i3i->value);
    }

    template <typename Archive>
    inline void
    load_construct_data(Archive& ar, I3Int* i3i,
                        unsigned)
    {
        int value;
        ar >> make_nvp("value", value);
        ::new(i3i) I3Int(value);
    }
}}
```

Non-default constructor: impl

```
namespace boost { namespace serialization {
    template <typename Archive>
    inline void
    save_construct_data(Archive& ar, const I3Int* i3i,
                        unsigned)
    {
        ar << make_nvp("value", i3i->value);
    }

    template <typename Archive>
    inline void
    load_construct_data(Archive& ar, I3Int* i3i,
                        unsigned)
    {
        int value;
        ar >> make_nvp("value", value);
        ::new(i3i) I3Int(value);
    }
}}
```

Non-default constructor: impl

```
namespace boost { namespace serialization {
    template <typename Archive>
    inline void
    save_construct_data(Archive& ar, const I3Int* i3i,
                        unsigned)
    {
        ar << make_nvp("value", i3i->value);
    }

    template <typename Archive>
    inline void
    load_construct_data(Archive& ar, I3Int* i3i,
                        unsigned)
    {
        int value;
        ar >> make_nvp("value", value);
        ::new(i3i) I3Int(value);
    }
}}
```

Non-default constructor: impl

```
namespace boost { namespace serialization {
    template <typename Archive>
    inline void
    save_construct_data(Archive& ar, const I3Int* i3i,
                        unsigned)
    {
        ar << make_nvp("value", i3i->value);
    }

    template <typename Archive>
    inline void
    load_construct_data(Archive& ar, I3Int* i3i,
                        unsigned)
    {
        int value;
        ar >> make_nvp("value", value);
        ::new(i3i) I3Int(value);
    }
}}
```

Non-default constructor: impl

```
namespace boost { namespace serialization {
    template <typename Archive>
    inline void
    save_construct_data(Archive& ar, const I3Int* i3i,
                        unsigned)
    {
        ar << make_nvp("value", i3i->value);
    }

    template <typename Archive>
    inline void
    load_construct_data(Archive& ar, I3Int* i3i,
                        unsigned)
    {
        int value;
        ar >> make_nvp("value", value);
        ::new(i3i) I3Int(value);
    }
}}
```

Non-default constructor: impl

```
namespace boost { namespace serialization {
    template <typename Archive>
    inline void
    save_construct_data(Archive& ar, const I3Int* i3i,
                        unsigned)
    {
        ar << make_nvp("value", i3i->value);
    }

    template <typename Archive>
    inline void
    load_construct_data(Archive& ar, I3Int* i3i,
                        unsigned)
    {
        int value;
        ar >> make_nvp("value", value);
        ::new(i3i) I3Int(value);
    }
}}
```

Non-default constructor: impl

```
namespace boost { namespace serialization {
    template <typename Archive>
    inline void
    save_construct_data(Archive& ar, const I3Int* i3i,
                        unsigned)
    {
        ar << make_nvp("value", i3i->value);
    }

    template <typename Archive>
    inline void
    load_construct_data(Archive& ar, I3Int* i3i,
                        unsigned)
    {
        int value;
        ar >> make_nvp("value", value);
        ::new(i3i) I3Int(value);
    }
}}
```

Versioning a class

```
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year

    ...
}
```

Versioning a class

```
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year

    ...
}
```

Versioning a class

```
struct I3Time
{
    uint8_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year

    ...
}
```

Versioning

```
#include <icetray/serialization.hpp>

BOOST_CLASS_VERSION( I3Time, 1);

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    ar & make_nvp("year", year);
    ar & make_nvp("daq_time", daq_time);

}
```

Versioning

```
#include <icetray/serialization.hpp>

BOOST_CLASS_VERSION( I3Time, 1);

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    ar & make_nvp("year", year);
    ar & make_nvp("daq_time", daq_time);

}

}
```

Versioning

```
#include <icetray/serialization.hpp>

BOOST_CLASS_VERSION( I3Time, 1);

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    ar & make_nvp("year", year);
    ar & make_nvp("daq_time", daq_time);

}
```

Versioning

```
#include <icetray/serialization.hpp>

BOOST_CLASS_VERSION( I3Time, 1);

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    if (typename Archive::is_loading() && version == 0)
    {
        uint32_t tmp;
        ar & make_nvp("year", tmp);
        year = tmp - 2000; // check for overflow?
    } else {
        ar & make_nvp("year", year);
    }
    ar & make_nvp("daqtime", daq_time);
}
```

Versioning

```
#include <icetray/serialization.hpp>

BOOST_CLASS_VERSION( I3Time, 1);

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    if (typename Archive::is_loading() && version == 0)
    {
        uint32_t tmp;
        ar & make_nvp("year", tmp);
        year = tmp - 2000; // check for overflow?
    } else {
        ar & make_nvp("year", year);
    }
    ar & make_nvp("daqtime", daq_time);
}
```

Versioning

```
#include <icetray/serialization.hpp>

BOOST_CLASS_VERSION( I3Time, 1); // version when saving

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    if (typename Archive::is_loading() && version == 0)
    {
        uint32_t tmp;
        ar & make_nvp("year", tmp);
        year = tmp - 2000; // check for overflow?
    } else {
        ar & make_nvp("year", year);
    }
    ar & make_nvp("daqtime", daq_time);
}
```

Versioning

```
#include <icetray/serialization.hpp>

BOOST_CLASS_VERSION( I3Time, 1);

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    if (typename Archive::is_loading() && version == 0)
    {
        uint32_t tmp;
        ar & make_nvp("year", tmp);
        year = tmp - 2000; // check for overflow?
    } else {
        ar & make_nvp("year", year);
    }
    ar & make_nvp("daqtime", daq_time);
}
```

Versioning

```
#include <icetray/serialization.hpp>

BOOST_CLASS_VERSION( I3Time, 1);

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    if (typename Archive::is_loading() && version == 0)
    {
        uint32_t tmp;
        ar & make_nvp("year", tmp);
        year = tmp - 2000; // check for overflow?
    } else {
        ar & make_nvp("year", year);
    }
    ar & make_nvp("daqtime", daq_time);
}
```

Versioning

```
#include <icetray/serialization.hpp>

BOOST_CLASS_VERSION( I3Time, 1);

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    if (typename Archive::is_loading() && version == 0)
    {
        uint32_t tmp;
        ar & make_nvp("year", tmp);
        year = tmp - 2000; // check for overflow?
    } else {
        ar & make_nvp("year", year);
    }
    ar & make_nvp("daqtime", daq_time);
}
```

Versioning

```
#include <icetray/serialization.hpp>

BOOST_CLASS_VERSION( I3Time, 1);

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    if (typename Archive::is_loading() && version == 0)
    {
        uint32_t tmp;
        ar & make_nvp("year", tmp);
        year = tmp - 2000; // check for overflow?
    } else {
        ar & make_nvp("year", year);
    }
    ar & make_nvp("daqtime", daq_time);
}
```

Versioning

```
#include <icetray/serialization.hpp>

BOOST_CLASS_VERSION( I3Time, 1);

template <typename Archive>
void
I3Time::serialize(Archive& ar, unsigned version)
{
    if (typename Archive::is_loading() && version == 0)
    {
        uint32_t tmp;
        ar & make_nvp("year", tmp);
        year = tmp - 2000; // check for overflow?
    } else {
        ar & make_nvp("year", year);
    }
    ar & make_nvp("daqtime", daq_time);
}
```

Splitting serialize() into save/load

```
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year

    template <typename Archive>
    void
    save(Archive& ar, unsigned version);

    template <typename Archive>
    void
    load(Archive& ar, unsigned version);

    BOOST_SERIALIZATION_SPLIT_MEMBER();
};
```

Splitting serialize() into save/load

```
struct I3Time
{
    uint32_t year; // year of event

    uint64_t daq_time; // tenths of nanoseconds
                        // since start of year

    template <typename Archive>
    void
    save(Archive& ar, unsigned version);

    template <typename Archive>
    void
    load(Archive& ar, unsigned version);

    BOOST_SERIALIZATION_SPLIT_MEMBER()
};
```

Splitting serialize() into save/load

```
template <typename Archive>
void
I3Time::save(Archive& ar, unsigned version)
{
    ar << make_nvp("year", year);
    ar << make_nvp("daq_time", daq_time);
}

template <typename Archive>
void
I3Time::load(Archive& ar, unsigned version)
{
    ...
}
```

Splitting serialize() into save/load

```
template <typename Archive>
void
I3Time::save(Archive& ar, unsigned version)
{
    ar << make_nvp("year", year);
    ar << make_nvp("daq_time", daq_time);
}

template <typename Archive>
void
I3Time::load(Archive& ar, unsigned version)
{
    ...
}
```

Splitting serialize() into save/load

```
template <typename Archive>
void
I3Time::save(Archive& ar, unsigned version)
{
    ar << make_nvp("year", year);
    ar << make_nvp("daq_time", daq_time);
}

template <typename Archive>
void
I3Time::load(Archive& ar, unsigned version)
{
    ...
}
```

Splitting serialize() into save/load

```
template <typename Archive>
void
I3Time::load(Archive& ar, unsigned version)
{
    if (version == 1)
        ar >> make_nvp("year", year);
    else
    {
        uint32_t tmp;
        ar >> make_nvp("year", tmp);
        year = tmp - 2000; // check for overflow?
    }
    ar >> make_nvp("daq_time", daq_time);
}
```

Versioning Gotcha

```
template <typename Archive>
void
I3Time::load(Archive& ar, unsigned version)
{
    if (version == 1)
        ar >> make_nvp("year", year);
    else
    {
        uint32_t tmp;
        ar >> make_nvp("year", tmp);
        year = tmp - 2000; // check for overflow?
    }
    ar >> make_nvp("daq_time", daq_time);
}
```

Versioning Gotcha

```
template <typename Archive>
void
I3Time::load(Archive& ar, unsigned version)
{
    if (version == 1)
        ar >> make_nvp("year", year);
    else
    {
        uint32_t tmp;
        ar >> make_nvp("year", tmp);
        year = tmp - 2000; // check for overflow?
    }
    ar >> make_nvp("daq_time", daq_time);
}
```

Versioning Gotcha

```
template <typename Archive>
void
I3Time::load(Archive& ar, unsigned version)
{
    if (version > 1)
        throw archive_exception(unsupported_version);
    else if (version == 1)
        ar >> make_nvp("year", year);
    else
    {
        uint32_t tmp;
        ar >> make_nvp("year", tmp);
        year = tmp - 2000; // check for overflow?
    }
    ar >> make_nvp("daq_time", daq_time);
}
```

Serializing the I3Frame itself

```
class I3Frame
{
    std::map<std::string, I3FrameObjectPtr> storage_;

public:

    template <typename Archive>
    void
    serialize(Archive& ar, const unsigned version)
    {
        ar & make_nvp("storage", storage_);
    }
};
```

Recap

- ▶ .i3 file format is just a series of serialized maps
- ▶ framework decoupled from data it handles
- ▶ reader/writer module simple
- ▶ you can easily filter the data
- ▶ LHS of map makes it easy do differentiate if e.g an I3Particle came from the LineFit reconstruction or elsewhere... users can develop their own naming schemes

Outline

`IceCube`

- `The physics`

- `The detector`

`IceTray`

`boost::serialization`

- `Lazy deserialization`

`boost::python`

`architectural python`

Dealing with catastrophe: the *lazy frame*

Disaster scenario: project *foo*

```
struct FooClass : I3FrameObject {
    // has serialize method and data
};

I3_POINTER_TYPEDEFS(FooClass);

class FooModule : public I3Module {
    void Physics(I3FramePtr)
    {
        // read some stuff from the frame and calculate some results
        FooClassPtr p(new FooClass);
        frame->Put("somewhere", p);
        PushFrame(frame);
    }
};

I3_MODULE(FooModule);
```

Disaster scenario: project *foo*

```
struct FooClass : I3FrameObject {
    // has serialize method and data
};

I3_POINTER_TYPEDEFS(FooClass);

class FooModule : public I3Module {
    void Physics(I3FramePtr)
    {
        // read some stuff from the frame and calculate some results
        FooClassPtr p(new FooClass);
        frame->Put("somewhere", p);
        PushFrame(frame);
    }
};

I3_MODULE(FooModule);
```

Disaster scenario: project *foo*

```
struct FooClass : I3FrameObject {
    // has serialize method and data
};

I3_POINTER_TYPEDEFS(FooClass);

class FooModule : public I3Module {
    void Physics(I3FramePtr)
    {
        // read some stuff from the frame and calculate some results
        FooClassPtr p(new FooClass);
        frame->Put("somewhere", p);
        PushFrame(frame);
    }
};

I3_MODULE(FooModule);
```

Disaster scenario: project *foo*

```
struct FooClass : I3FrameObject {
    // has serialize method and data
};

I3_POINTER_TYPEDEFS(FooClass);

class FooModule : public I3Module {
    void Physics(I3FramePtr)
    {
        // read some stuff from the frame and calculate some results
        FooClassPtr p(new FooClass);
        frame->Put("somewhere", p);
        PushFrame(frame);
    }
};

I3_MODULE(FooModule);
```

Disaster scenario: project *foo*

```
struct FooClass : I3FrameObject {
    // has serialize method and data
};

I3_POINTER_TYPEDEFS(FooClass);

class FooModule : public I3Module {
    void Physics(I3FramePtr)
    {
        // read some stuff from the frame and calculate some results
        FooClassPtr p(new FooClass);
        frame->Put("somewhere", p);
        PushFrame(frame);
    }
};

I3_MODULE(FooModule);
```

Disaster scenario: project *foo*

```
struct FooClass : I3FrameObject {
    // has serialize method and data
};

I3_POINTER_TYPEDEFS(FooClass);

class FooModule : public I3Module {
    void Physics(I3FramePtr)
    {
        // read some stuff from the frame and calculate some results
        FooClassPtr p(new FooClass);
        frame->Put("somewhere", p);
        PushFrame(frame);
    }
};

I3_MODULE(FooModule);
```

Disaster scenario: project *foo*

```
struct FooClass : I3FrameObject {
    // has serialize method and data
};

I3_POINTER_TYPEDEFS(FooClass);

class FooModule : public I3Module {
    void Physics(I3FramePtr)
    {
        // read some stuff from the frame and calculate some results
        FooClassPtr p(new FooClass);
        frame->Put("somewhere", p);
PushFrame(frame);
    }
};

I3_MODULE(FooModule);
```

Writing a contaminated file

```
from icecube import icetray, dataio, foo

tray.AddModule("I3Reader", "reader",
               Filename="in.i3")

tray.AddModule("FooModule", "foo")

tray.AddModule("I3Writer", "writer",
               Filename="contaminated.i3")
```

Writing a contaminated file

```
from icecube import icetray, dataio  
  
tray.AddModule("I3Reader", "reader",  
    Filename="contaminated.i3")
```

Writing a contaminated file

```
from icecube import icetray, dataio  
  
tray.AddModule("I3Reader", "reader",  
    Filename="contaminated.i3")
```

boost::archive_error::unregistered_class

I3Frame two-step serialization

```
template <typename OArchive>
void
I3Frame::save(OArchive& disk_ar, const unsigned version)
{
    std::map<std::string, std::vector<char> > disk_storage;
    BOOST_FOREACH (storage_t::iterator iter, storage_)
    {
        vector<char>& buf = disk_storage[iter->first];
        io::stream<io::back_insert_device<vector<char> > >
            vs(buf);
        {
            OArchive mem_ar(vs);
            mem_ar << iter->second;
        }
    }
    disk_ar << disk_storage;
}
```

I3Frame two-step serialization

```
template <typename OArchive>
void
I3Frame::save(OArchive& disk_ar, const unsigned version)
{
    std::map<std::string, std::vector<char> > disk_storage;
    BOOST_FOREACH (storage_t::iterator iter, storage_)
    {
        vector<char>& buf = disk_storage[iter->first];
        io::stream<io::back_insert_device<vector<char> > >
            vs(buf);
        {
            OArchive mem_ar(vs);
            mem_ar << iter->second;
        }
    }
    disk_ar << disk_storage;
}
```

I3Frame two-step serialization

```
template <typename OArchive>
void
I3Frame::save(OArchive& disk_ar, const unsigned version)
{
    std::map<std::string, std::vector<char> > disk_storage;
BOOST_FOREACH (storage_t::iterator iter, storage_)
{
    vector<char>& buf = disk_storage[iter->first];
    io::stream<io::back_insert_device<vector<char> > >
        vs(buf);
    {
        OArchive mem_ar(vs);
        mem_ar << iter->second;
    }
}
disk_ar << disk_storage;
}
```

I3Frame two-step serialization

```
template <typename OArchive>
void
I3Frame::save(OArchive& disk_ar, const unsigned version)
{
    std::map<std::string, std::vector<char> > disk_storage;
    BOOST_FOREACH (storage_t::iterator iter, storage_)
    {
        vector<char>& buf = disk_storage[iter->first];
        io::stream<io::back_insert_device<vector<char> > >
            vs(buf);
        {
            OArchive mem_ar(vs);
            mem_ar << iter->second;
        }
    }
    disk_ar << disk_storage;
}
```

I3Frame two-step serialization

```
template <typename OArchive>
void
I3Frame::save(OArchive& disk_ar, const unsigned version)
{
    std::map<std::string, std::vector<char> > disk_storage;
    BOOST_FOREACH (storage_t::iterator iter, storage_)
    {
        vector<char>& buf = disk_storage[iter->first];
        io::stream<io::back_insert_device<vector<char> > >
        vs(buf);
        {
            OArchive mem_ar(vs);
            mem_ar << iter->second;
        }
    }
    disk_ar << disk_storage;
}
```

I3Frame two-step serialization

```
template <typename OArchive>
void
I3Frame::save(OArchive& disk_ar, const unsigned version)
{
    std::map<std::string, std::vector<char> > disk_storage;
    BOOST_FOREACH (storage_t::iterator iter, storage_)
    {
        vector<char>& buf = disk_storage[iter->first];
        io::stream<io::back_insert_device<vector<char> > >
            vs(buf);
        {
            OArchive mem_ar(vs);
            mem_ar << iter->second;
        }
    }
    disk_ar << disk_storage;
}
```

I3Frame two-step serialization

```
template <typename OArchive>
void
I3Frame::save(OArchive& disk_ar, const unsigned version)
{
    std::map<std::string, std::vector<char> > disk_storage;
    BOOST_FOREACH (storage_t::iterator iter, storage_)
    {
        vector<char>& buf = disk_storage[iter->first];
        io::stream<io::back_insert_device<vector<char> > >
            vs(buf);
        {
            OArchive mem_ar(vs);
            mem_ar << iter->second;
        }
    }
    disk_ar << disk_storage;
}
```

I3Frame two-step serialization

```
template <typename OArchive>
void
I3Frame::save(OArchive& disk_ar, const unsigned version)
{
    std::map<std::string, std::vector<char> > disk_storage;
    BOOST_FOREACH (storage_t::iterator iter, storage_)
    {
        vector<char>& buf = disk_storage[iter->first];
        io::stream<io::back_insert_device<vector<char> > >
            vs(buf);
        {
            OArchive mem_ar(vs);
            mem_ar << iter->second;
        }
    }
    disk_ar << disk_storage;
}
```

What you get

Good:

- ▶ You can only report "exception 'unregistered_class' caught when loading frameobject at slot *someplace*"
- ▶ You can skip unregistered, lost, or corrupt classes
- ▶ You can write checksums along with the buffers if you like
- ▶ Works with any boost archive and any iostream type

Bad:

- ▶ You only report "exception 'unregistered_class' caught when loading frameobject at slot *someplace*". You don't know what the type might be or where to find it.
- ▶ You pay for deserializing everything in the frame, every time.

What you get

Good:

- ▶ You can only report "exception 'unregistered_class' caught when loading frameobject at slot *someplace*"
- ▶ **You can skip unregistered, lost, or corrupt classes**
- ▶ You can write checksums along with the buffers if you like
- ▶ Works with any boost archive and any iostream type

Bad:

- ▶ You only report "exception 'unregistered_class' caught when loading frameobject at slot *someplace*". You don't know what the type might be or where to find it.
- ▶ You pay for deserializing everything in the frame, every time.

What you get

Good:

- ▶ You can only report "exception 'unregistered_class' caught when loading frameobject at slot *someplace*"
- ▶ You can skip unregistered, lost, or corrupt classes
- ▶ **You can write checksums along with the buffers if you like**
- ▶ Works with any boost archive and any iostream type

Bad:

- ▶ You only report "exception 'unregistered_class' caught when loading frameobject at slot *someplace*". You don't know what the type might be or where to find it.
- ▶ You pay for deserializing everything in the frame, every time.

What you get

Good:

- ▶ You can only report "exception 'unregistered_class' caught when loading frameobject at slot *someplace*"
- ▶ You can skip unregistered, lost, or corrupt classes
- ▶ You can write checksums along with the buffers if you like
- ▶ **Works with any boost archive and any iostream type**

Bad:

- ▶ You only report "exception 'unregistered_class' caught when loading frameobject at slot *someplace*". You don't know what the type might be or where to find it.
- ▶ You pay for deserializing everything in the frame, every time.

What you get

Good:

- ▶ You can only report "exception 'unregistered_class' caught when loading frameobject at slot *someplace*"
- ▶ You can skip unregistered, lost, or corrupt classes
- ▶ You can write checksums along with the buffers if you like
- ▶ Works with any boost archive and any iostream type

Bad:

- ▶ You only report "exception 'unregistered_class' caught when loading frameobject at slot *someplace*". You don't know what the type might be or where to find it.
- ▶ You pay for deserializing everything in the frame, every time.

What you get

Good:

- ▶ You can only report "exception 'unregistered_class' caught when loading frameobject at slot *someplace*"
- ▶ You can skip unregistered, lost, or corrupt classes
- ▶ You can write checksums along with the buffers if you like
- ▶ Works with any boost archive and any iostream type

Bad:

- ▶ You only report "exception 'unregistered_class' caught when loading frameobject at slot *someplace*". You don't know what the type might be or where to find it.
- ▶ You pay for deserializing everything in the frame, every time.

What you get

Good:

- ▶ You can only report "exception 'unregistered_class' caught when loading frameobject at slot *someplace*"
- ▶ You can skip unregistered, lost, or corrupt classes
- ▶ You can write checksums along with the buffers if you like
- ▶ Works with any boost archive and any iostream type

Bad:

- ▶ You only report "exception 'unregistered_class' caught when loading frameobject at slot *someplace*". You don't know what the type might be or where to find it.
- ▶ You pay for deserializing everything in the frame, every time.

Datastructure for a lazily-deserializing frame

```
struct value_t
{
    mutable std::string type_name;
    mutable std::vector<char> buf;
    I3FrameObjectConstPtr obj_p;
};

typedef map<string, shared_ptr<value_t> > storage_t;

class I3Frame {
    storage_t storage_;
    // ...
};
```

Datastructure for a lazily-deserializing frame

```
struct value_t
{
    mutable std::string type_name;
    mutable std::vector<char> buf;
    I3FrameObjectConstPtr obj_p;
};

typedef map<string, shared_ptr<value_t> > storage_t;

class I3Frame {
    storage_t storage_;
    // ...
};
```

Datastructure for a lazily-deserializing frame

```
struct value_t
{
    mutable std::string type_name;
    mutable std::vector<char> buf;
    I3FrameObjectConstPtr obj_p;
};

typedef map<string, shared_ptr<value_t> > storage_t;

class I3Frame {
    storage_t storage_;
    // ...
};
```

Datastructure for a lazily-deserializing frame

```
struct value_t
{
    mutable std::string type_name;
    mutable std::vector<char> buf;
    I3FrameObjectConstPtr obj_p;
};

typedef map<string, shared_ptr<value_t> > storage_t;

class I3Frame {
    storage_t storage_;
    // ...
};
```

Datastructure for a lazily-deserializing frame

```
struct value_t
{
    mutable std::string type_name;
mutable std::vector<char> buf;
    I3FrameObjectConstPtr obj_p;
};

typedef map<string, shared_ptr<value_t> > storage_t;

class I3Frame {
    storage_t storage_;
    // ...
};
```

Datastructure for a lazily-deserializing frame

```
struct value_t
{
    mutable std::string type_name;
    mutable std::vector<char> buf;
I3FrameObjectConstPtr obj_p;
};

typedef map<string, shared_ptr<value_t> > storage_t;

class I3Frame {
    storage_t storage_;
    // ...
};
```

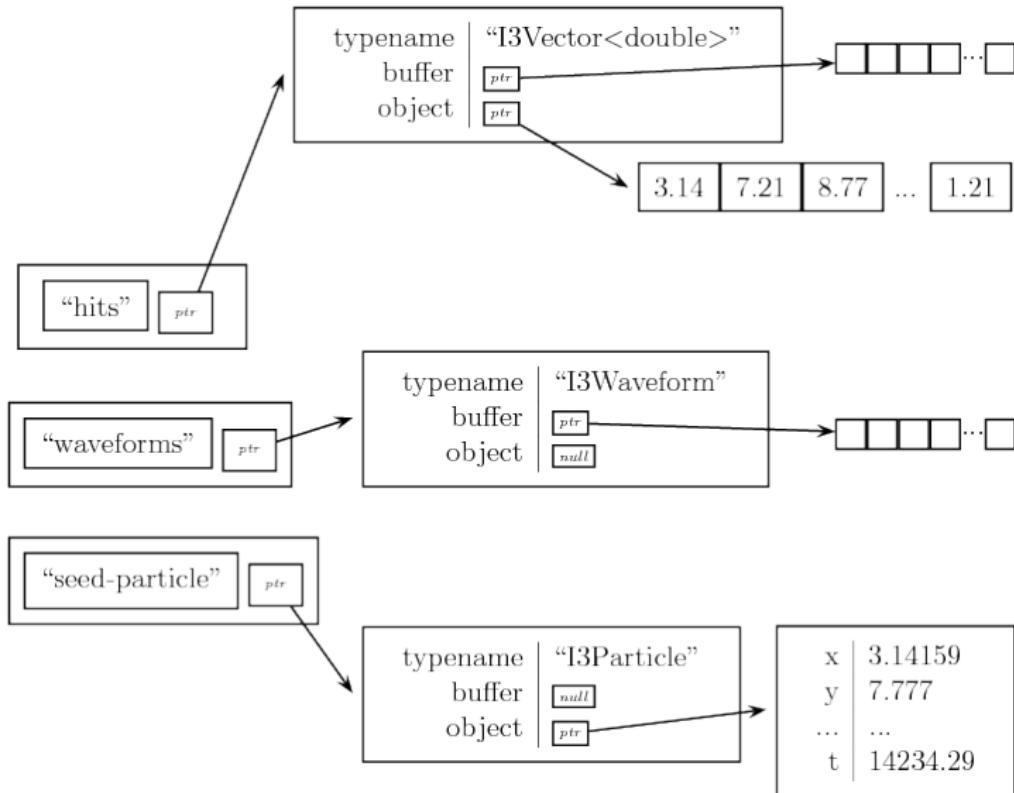
Datastructure for a lazily-deserializing frame

```
struct value_t
{
    mutable std::string type_name;
    mutable std::vector<char> buf;
    I3FrameObjectConstPtr obj_p;
};

typedef map<string, shared_ptr<value_t> > storage_t;

class I3Frame {
    storage_t storage_;
    // ...
};
```

Internals of the lazy frame



Lazy serialization

```
template <typename Archive>
void
value_t::save(Archive& ar, const unsigned version)
{
    if (type_name.size() == 0)
        type_name = name_of(obj_p);
    ar << type_name;
    if (buf.size() == 0) {
        io::stream<io::back_insert_device<vector<char> > >
            vs(buf);
        {
            Archive buf_archive(vs);
            buf_archive << obj_p;
        }
        vs.flush();
    }
    ar << buf;
}
```

Lazy serialization

```
template <typename Archive>
void
value_t::save(Archive& ar, const unsigned version)
{
    if (type_name.size() == 0)
        type_name = name_of(obj_p);
    ar << type_name;
    if (buf.size() == 0) {
        io::stream<io::back_insert_device<vector<char> > >
            vs(buf);
        {
            Archive buf_archive(vs);
            buf_archive << obj_p;
        }
        vs.flush();
    }
    ar << buf;
}
```

Lazy serialization

```
template <typename Archive>
void
value_t::save(Archive& ar, const unsigned version)
{
    if (type_name.size() == 0)
        type_name = name_of(obj_p);
    ar << type_name;
    if (buf.size() == 0) {
        io::stream<io::back_insert_device<vector<char> > >
            vs(buf);
        {
            Archive buf_archive(vs);
            buf_archive << obj_p;
        }
        vs.flush();
    }
    ar << buf;
}
```

Lazy serialization

```
template <typename Archive>
void
value_t::save(Archive& ar, const unsigned version)
{
    if (type_name.size() == 0)
        type_name = name_of(obj_p);
    ar << type_name;
    if (buf.size() == 0) {
        io::stream<io::back_insert_device<vector<char> > >
            vs(buf);
        {
            Archive buf_archive(vs);
            buf_archive << obj_p;
        }
        vs.flush();
    }
    ar << buf;
}
```

Lazy serialization

```
template <typename Archive>
void
value_t::save(Archive& ar, const unsigned version)
{
    if (type_name.size() == 0)
        type_name = name_of(obj_p);
    ar << type_name;
    if (buf.size() == 0) {
        io::stream<io::back_insert_device<vector<char> > >
            vs(buf);
        {
            Archive buf_archive(vs);
            buf_archive << obj_p;
        }
        vs.flush();
    }
ar << buf;
}
```

Lazy iserialization

```
template <typename Archive>
void
value_t::load(Archive& ar, const unsigned version)
{
    ar >> type_name;
    ar >> buf;
    obj_p.reset();
}
```

Lazy serialization

```
template <typename Archive>
void
value_t::load(Archive& ar, const unsigned version)
{
    ar >> type_name;
    ar >> buf;
    obj_p.reset();
}
```

Lazy iserialization

```
template <typename Archive>
void
value_t::load(Archive& ar, const unsigned version)
{
    ar >> type_name;
ar >> buf;
    obj_p.reset();
}
```

Lazy iserialization

```
template <typename Archive>
void
value_t::load(Archive& ar, const unsigned version)
{
    ar >> type_name;
    ar >> buf;
    obj_p.reset();
}
```

I3Frame two-step serialization

```
template <typename OArchive>
void
I3Frame::serialize(OArchive& ar, const unsigned version)
{
    ar & storage_;
}
```

- ▶ If something goes wrong, we can report the name, typename, and size of the problematic object
- ▶ Can even pass objects we don't have code for completely through a tray if we don't touch them
- ▶ Only serialize/deserialize when necessary

dataio-shovel

boost::python

Hello World

```
#include <boost/python.hpp>
using namespace boost::python;

void hello_world() { std::cout << "hello world\n"; }

BOOST_PYTHON_MODULE(mymodule)
{
    std::cout << "Loading mymodule\n";
    def("hello_world", hello_world);
}

>>> import mymodule
Loading mymodule
>>> mymodule.hello_world
<Boost.Python.function object at 0xc0b5c0>
>>> mymodule.hello_world()
hello world
```

Hello World

```
#include <boost/python.hpp>
using namespace boost::python;

void hello_world() { std::cout << "hello world\n"; }

BOOST_PYTHON_MODULE(mymodule)
{
    std::cout << "Loading mymodule\n";
    def("hello_world", hello_world);
}

>>> import mymodule
Loading mymodule
>>> mymodule.hello_world
<Boost.Python.function object at 0xc0b5c0>
>>> mymodule.hello_world()
hello world
```

Hello World

```
#include <boost/python.hpp>
using namespace boost::python;

void hello_world() { std::cout << "hello world\n"; }

BOOST_PYTHON_MODULE(mymodule)
{
    std::cout << "Loading mymodule\n";
    def("hello_world", hello_world);
}

>>> import mymodule
Loading mymodule
>>> mymodule.hello_world
<Boost.Python.function object at 0xc0b5c0>
>>> mymodule.hello_world()
hello world
```

Hello World

```
#include <boost/python.hpp>
using namespace boost::python;

void hello_world() { std::cout << "hello world\n"; }
// extern "C" void initmymodule()...
BOOST_PYTHON_MODULE(mymodule)
{
    std::cout << "Loading mymodule\n";
    def("hello_world", hello_world);
}

>>> import mymodule
Loading mymodule
>>> mymodule.hello_world
<Boost.Python.function object at 0xc0b5c0>
>>> mymodule.hello_world()
hello world
```

Hello World

```
#include <boost/python.hpp>
using namespace boost::python;

void hello_world() { std::cout << "hello world\n"; }

BOOST_PYTHON_MODULE(mymodule)
{
    std::cout << "Loading mymodule\n";
    def("hello_world", hello_world);
}

>>> import mymodule
Loading mymodule
>>> mymodule.hello_world
<Boost.Python.function object at 0xc0b5c0>
>>> mymodule.hello_world()
hello world
```

Hello World

```
#include <boost/python.hpp>
using namespace boost::python;

void hello_world() { std::cout << "hello world\n"; }

BOOST_PYTHON_MODULE(mymodule)
{
    std::cout << "Loading mymodule\n";
    def("hello_world", hello_world);
}

>>> import mymodule
Loading mymodule
>>> mymodule.hello_world
<Boost.Python.function object at 0xc0b5c0>
>>> mymodule.hello_world()
hello world
```

Hello World

```
#include <boost/python.hpp>
using namespace boost::python;

void hello_world() { std::cout << "hello world\n"; }

BOOST_PYTHON_MODULE(mymodule)
{
    std::cout << "Loading mymodule\n";
    def("hello_world", hello_world);
}

>>> import mymodule
Loading mymodule
>>> mymodule.hello_world
<Boost.Python.function object at 0xc0b5c0>
>>> mymodule.hello_world()
hello world
```

Hello World

```
#include <boost/python.hpp>
using namespace boost::python;

void hello_world() { std::cout << "hello world\n"; }

BOOST_PYTHON_MODULE(mymodule)
{
    std::cout << "Loading mymodule\n";
    def("hello_world", hello_world);
}

>>> import mymodule
Loading mymodule
>>> mymodule.hello_world
<Boost.Python.function object at 0xc0b5c0>
>>> mymodule.hello_world()
hello world
```

Hello World

```
#include <boost/python.hpp>
using namespace boost::python;

void hello_world() { std::cout << "hello world\n"; }

BOOST_PYTHON_MODULE(mymodule)
{
    std::cout << "Loading mymodule\n";
    def("hello_world", hello_world);
}

>>> import mymodule
Loading mymodule
>>> mymodule.hello_world
<Boost.Python.function object at 0xc0b5c0>
>>> mymodule.hello_world()
hello world
```

Hello World

```
#include <boost/python.hpp>
using namespace boost::python;

void hello_world() { std::cout << "hello world\n"; }

BOOST_PYTHON_MODULE(mymodule)
{
    std::cout << "Loading mymodule\n";
    def("hello_world", hello_world);
}

>>> import mymodule
Loading mymodule
>>> mymodule.hello_world
<Boost.Python.function object at 0xc0b5c0>
>>> mymodule.hello_world()
hello world
```

Classes

```
struct S {
    void hello_world() { std::cout << "hello world\n"; }
};

BOOST_PYTHON_MODULE(mymodule)
{
    class_<S>("S")
        .def("hello_world", hello_world)
    ;
}
}

>>> import mymodule
>>> mymodule.S
<class 'mymodule.S'>
>>> mymodule.S()
<mymodule.S object at 0x20dbdb8>
>>> s.hello_world()
hello world
```

Classes

```
struct S {
    void hello_world() { std::cout << "hello world\n"; }
};

BOOST_PYTHON_MODULE(mymodule)
{
    class_<S>("S")
        .def("hello_world", hello_world)
    ;
}
}

>>> import mymodule
>>> mymodule.S
<class 'mymodule.S'>
>>> mymodule.S()
<mymodule.S object at 0x20dbdb8>
>>> s.hello_world()
hello world
```

Classes

```
struct S {  
    void hello_world() { std::cout << "hello world\n"; }  
};  
  
BOOST_PYTHON_MODULE(mymodule)  
{  
    class_<S>("S")  
        .def("hello_world", hello_world)  
        ;  
}  
  
>>> import mymodule  
>>> mymodule.S  
<class 'mymodule.S'>  
>>> mymodule.S()  
<mymodule.S object at 0x20dbdb8>  
>>> s.hello_world()  
hello world
```

Classes

```
struct S {
    void hello_world() { std::cout << "hello world\n"; }
};

BOOST_PYTHON_MODULE(mymodule)
{
    class_<S>("S")
        .def("hello_world", hello_world)
        ;
}

>>> import mymodule
>>> mymodule.S
<class 'mymodule.S'>
>>> mymodule.S()
<mymodule.S object at 0x20dbdb8>
>>> s.hello_world()
hello world
```

Classes

```
struct S {  
    void hello_world() { std::cout << "hello world\n"; }  
};  
  
BOOST_PYTHON_MODULE(mymodule)  
{  
    class_<S>("S")  
        .def("hello_world", hello_world)  
        ;  
}  
  
>>> import mymodule  
>>> mymodule.S  
<class 'mymodule.S'>  
>>> mymodule.S()  
<mymodule.S object at 0x20dbdb8>  
>>> s.hello_world()  
hello world
```

Classes

```
struct S {  
    void hello_world() { std::cout << "hello world\n"; }  
};  
  
BOOST_PYTHON_MODULE(mymodule)  
{  
    class_<S>("S")  
        .def("hello_world", hello_world)  
        ;  
}  
  
>>> import mymodule  
>>> mymodule.S  
<class 'mymodule.S'>  
>>> mymodule.S()  
<mymodule.S object at 0x20dbdb8>  
>>> s.hello_world()  
hello world
```

Classes

```
struct S {  
    void hello_world() { std::cout << "hello world\n"; }  
};  
  
BOOST_PYTHON_MODULE(mymodule)  
{  
    class_<S>("S")  
        .def("hello_world", hello_world)  
        ;  
}  
  
>>> import mymodule  
>>> mymodule.S  
<class 'mymodule.S'>  
>>> mymodule.S()  
<mymodule.S object at 0x20dbdb8>  
>>> s.hello_world()  
hello world
```

Classes

```
struct S {  
    void hello_world() { std::cout << "hello world\n"; }  
};  
  
BOOST_PYTHON_MODULE(mymodule)  
{  
    class_<S>("S")  
        .def("hello_world", hello_world)  
        ;  
}  
  
>>> import mymodule  
>>> mymodule.S  
<class 'mymodule.S'>  
>>> mymodule.S()  
<mymodule.S object at 0x20dbdb8>  
>>> s.hello_world()  
hello world
```

Classes

```
struct S {  
    void hello_world() { std::cout << "hello world\n"; }  
};  
  
BOOST_PYTHON_MODULE(mymodule)  
{  
    class_<S>("S")  
        .def("hello_world", hello_world)  
        ;  
}  
  
>>> import mymodule  
>>> mymodule.S  
<class 'mymodule.S'>  
>>> mymodule.S()  
<mymodule.S object at 0x20dbdb8>  
>>> s.hello_world()  
hello world
```

Data Members

```
struct I3Int
{
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int()
>>> i.value
0
>>> i.value = 13
>>> i.value
13
```

Data Members

```
struct I3Int
{
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int()
>>> i.value
0
>>> i.value = 13
>>> i.value
13
```

Data Members

```
struct I3Int
{
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int()
>>> i.value
0
>>> i.value = 13
>>> i.value
13
```

Data Members

```
struct I3Int
{
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int()
>>> i.value
0
>>> i.value = 13
>>> i.value
13
```

Data Members

```
struct I3Int
{
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int()
>>> i.value
0
>>> i.value = 13
>>> i.value
13
```

Data Members

```
struct I3Int
{
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int()
>>> i.value
0
>>> i.value = 13
>>> i.value
13
```

Data Members

```
struct I3Int
{
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int()
>>> i.value
0
>>> i.value = 13
>>> i.value
13
```

Data Members

```
struct I3Int
{
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int()
>>> i.value
0
>>> i.value = 13
>>> i.value
13
```

Data Members

```
struct I3Int
{
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int()
>>> i.value
0
>>> i.value = 13
>>> i.value
13
```

Data Members

```
struct I3Int
{
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int()
>>> i.value
0
>>> i.value = 13
>>> i.value
13
```

Data Members

```
struct I3Int
{
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int()
>>> i.value
0
>>> i.value = 13
>>> i.value
13
```

Data Members

```
struct I3Int
{
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int()
>>> i.value
0
>>> i.value = 13
>>> i.value
13
```

Constructors

```
struct I3Int
{
    I3Int(int i) : value(i) { }
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def(init<int>())
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int(777)
>>> i.value
777
```

Constructors

```
struct I3Int
{
    I3Int(int i) : value(i) { }
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def(init<int>())
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int(777)
>>> i.value
777
```

Constructors

```
struct I3Int
{
    I3Int(int i) : value(i) { }
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def(init<int>())
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int(777)
>>> i.value
777
```

Constructors

```
struct I3Int
{
    I3Int(int i) : value(i) { }
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def(init<int>())
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int(777)
>>> i.value
777
```

Constructors

```
struct I3Int
{
    I3Int(int i) : value(i) { }
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def(init<int>())
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int(777)
>>> i.value
777
```

Constructors

```
struct I3Int
{
    I3Int(int i) : value(i) { }
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def(init<int>())
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int(777)
>>> i.value
777
```

No default constructor

```
struct I3Int
{
    I3Int(int i) : value(i) { }
    int value;
private:
    I3Int();
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int", init<int>())
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int(777)
>>> i.value
777
```

No default constructor

```
struct I3Int
{
    I3Int(int i) : value(i) { }
    int value;
private:
    I3Int();
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int", init<int>())
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int(777)
>>> i.value
777
```

No default constructor

```
struct I3Int
{
    I3Int(int i) : value(i) { }
    int value;
private:
I3Int();
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int", init<int>())
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int(777)
>>> i.value
777
```

No default constructor

```
struct I3Int
{
    I3Int(int i) : value(i) { }
    int value;
private:
    I3Int();
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int", init<int>())
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int(777)
>>> i.value
777
```

No default constructor

```
struct I3Int
{
    I3Int(int i) : value(i) { }
    int value;
private:
    I3Int();
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int", init<int>())
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int(777)
>>> i.value
777
```

No default constructor

```
struct I3Int
{
    I3Int(int i) : value(i) { }
    int value;
private:
    I3Int();
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int", init<int>())
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> from icecube import icetray
>>> i = icetray.I3Int(777)
>>> i.value
777
```

Abstract base class

```
struct I3FrameObject
{
    virtual ~I3FrameObject();
    // and a serialize method
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3FrameObject>("I3FrameObject",
        "Base class for all objects that live in the I3Frame",
        no_init)
    ;
}
```

Abstract base class

```
struct I3FrameObject
{
    virtual ~I3FrameObject();
    // and a serialize method
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3FrameObject>("I3FrameObject",
        "Base class for all objects that live in the I3Frame",
        no_init)
    ;
}
```

Abstract base class

```
struct I3FrameObject
{
    virtual ~I3FrameObject();
    // and a serialize method
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3FrameObject>("I3FrameObject",
        "Base class for all objects that live in the I3Frame",
        no_init)
    ;
}
```

Abstract base class

```
struct I3FrameObject
{
    virtual ~I3FrameObject();
    // and a serialize method
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3FrameObject>("I3FrameObject",
        "Base class for all objects that live in the I3Frame",
        no_init)
    ;
}
```

Abstract base class

```
struct I3FrameObject
{
    virtual ~I3FrameObject();
    // and a serialize method
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3FrameObject>("I3FrameObject",
        "Base class for all objects that live in the I3Frame",
        no_init)
    ;
}
```

Abstract base class

```
struct I3FrameObject
{
    virtual ~I3FrameObject();
    // and a serialize method
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3FrameObject>("I3FrameObject",
        "Base class for all objects that live in the I3Frame",
        no_init)
    ;
}

>>> from icecube import icetray
>>>
```

Abstract base class

```
struct I3FrameObject
{
    virtual ~I3FrameObject();
    // and a serialize method
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3FrameObject>("I3FrameObject",
        "Base class for all objects that live in the I3Frame",
        no_init)
    ;
}

>>> from icecube import icetray
>>> obj = icetray.I3FrameObject()
```

Abstract base class

```
struct I3FrameObject
{
    virtual ~I3FrameObject();
    // and a serialize method
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3FrameObject>("I3FrameObject",
        "Base class for all objects that live in the I3Frame",
        no_init)
    ;
}

>>> from icecube import icetray
>>> obj = icetray.I3FrameObject()
RuntimeError: This class cannot be instantiated from Python
```

Base classes

```
struct I3Int : I3FrameObject
{
    virtual ~I3Int();
    int value;
};

void print_address(I3FrameObject& f)
{
    std::cout << "addy is " << &f << "\n";
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
        ;
    def("print_address", print_address);
}
```

Base classes

```
struct I3Int : I3FrameObject
{
    virtual ~I3Int();
    int value;
};

void print_address(I3FrameObject& f)
{
    std::cout << "addy is " << &f << "\n";
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
        ;
    def("print_address", print_address);
}
```

Base classes

```
struct I3Int : I3FrameObject
{
    virtual ~I3Int();
    int value;
};

void print_address(I3FrameObject& f)
{
    std::cout << "addy is " << &f << "\n";
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
        ;
    def("print_address", print_address);
}
```

Base classes

```
struct I3Int : I3FrameObject
{
    virtual ~I3Int();
    int value;
};

void print_address(I3FrameObject& f)
{
    std::cout << "addy is " << &f << "\n";
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
        ;
    def("print_address", print_address);
}
```

Base classes

```
struct I3Int : I3FrameObject
{
    virtual ~I3Int();
    int value;
};

void print_address(I3FrameObject& f)
{
    std::cout << "addy is " << &f << "\n";
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
        ;
    def("print_address", print_address);
}
```

Base classes

```
struct I3Int : I3FrameObject
{
    virtual ~I3Int();
    int value;
};

void print_address(I3FrameObject& f)
{
    std::cout << "addy is " << &f << "\n";
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
        ;
    def("print_address", print_address);
}
```

Ooops

```
>>> from icecube import icetray  
>>>
```

Ooops

```
>>> from icecube import icetray  
>>> i = icetray.I3Int()  
>>>
```

Ooops

```
>>> from icecube import icetray  
>>> i = icetray.I3Int()  
>>> icetray.print_address(i)
```

Oops

```
>>> from icecube import icetray
>>> i = icetray.I3Int()
>>> icetray.print_address(i)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
Boost.Python.ArgumentError: Python argument types in
    icecube.icetray.print_address(I3Int)
did not match C++ signature:
    print_address(I3FrameObject {lvalue})
```

Ooops

```
>>> from icecube import icetray
>>> i = icetray.I3Int()
>>> icetray.print_address(i)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
Boost.Python.ArgumentError: Python argument types in
    icecube.icetray.print_address(I3Int)
did not match C++ signature:
    print_address(I3FrameObject {lvalue})
```

Ooops

```
>>> from icecube import icetray
>>> i = icetray.I3Int()
>>> icetray.print_address(i)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
Boost.Python.ArgumentError: Python argument types in
  icecube.icetray.print_address(I3Int)
did not match C++ signature:
    print_address(I3FrameObject {lvalue})
```

Base classes

```
struct I3Int : I3FrameObject
{
    virtual ~I3Int();
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int>("I3Int")
        .def_readwrite("value", &I3Int::value)
    ;
}
```

Base classes

```
struct I3Int : I3FrameObject
{
    virtual ~I3Int();
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_bases<I3FrameObject> >("I3Int")
        .def_readwrite("value", &I3Int::value)
    ;
}
```

Base classes

```
struct I3Int : I3FrameObject
{
    virtual ~I3Int();
    int value;
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int, bases<I3FrameObject>>("I3Int")
        .def_readwrite("value", &I3Int::value)
    ;
}

>>> i = icetray.I3Int()
>>> icetray.print_address(i)
addy is 0xe99e50
```

Held type

```
I3IntPtr make_int(int i)
{
    return I3IntPtr(new I3Int(i));
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int, bases<I3FrameObject>>("I3Int")
        .def_readwrite("value", &I3Int::value)
        ;
    def("make_int", make_int);
}
```

Held type

```
I3IntPtr make_int(int i)
{
    return I3IntPtr(new I3Int(i));
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int, bases<I3FrameObject>>("I3Int")
        .def_readwrite("value", &I3Int::value)
        ;
    def("make_int", make_int);
}
```

Held type

```
I3IntPtr make_int(int i)
{
    return I3IntPtr(new I3Int(i));
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int, bases<I3FrameObject>>("I3Int")
        .def_readwrite("value", &I3Int::value)
        ;
    def("make_int", make_int);
}
```

Held type

```
I3IntPtr make_int(int i) // boost::shared_ptr<I3Int>
{
    return I3IntPtr(new I3Int(i));
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int, bases<I3FrameObject>>("I3Int")
        .def_readwrite("value", &I3Int::value)
        ;
    def("make_int", make_int);
}
```

Held type

```
I3IntPtr make_int(int i)
{
    return I3IntPtr(new I3Int(i));
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int, bases<I3FrameObject> >("I3Int")
        .def_readwrite("value", &I3Int::value)
        ;
    def("make_int", make_int);
}
```

Held type

```
I3IntPtr make_int(int i)
{
    return I3IntPtr(new I3Int(i));
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int, bases<I3FrameObject> >("I3Int")
        .def_readwrite("value", &I3Int::value)
        ;
    def("make_int", make_int);
}

>>> i = icetray.make_int(777)
```

Held type

```
I3IntPtr make_int(int i)
{
    return I3IntPtr(new I3Int(i));
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int, bases<I3FrameObject> >("I3Int")
        .def_readwrite("value", &I3Int::value)
        ;
    def("make_int", make_int);
}

>>> i = icetray.make_int(777)
TypeError: No to_python (by-value) converter found
          for C++ type: boost::shared_ptr<I3Int>
```

Held type

```
I3IntPtr make_int(int i)
{
    return I3IntPtr(new I3Int(i));
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int, bases<I3FrameObject>, I3IntPtr >("I3Int")
        .def_readwrite("value", &I3Int::value)
        ;
    def("make_int", make_int);
}

>>> i = icetray.make_int(777)
```

Held type

```
I3IntPtr make_int(int i)
{
    return I3IntPtr(new I3Int(i));
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Int, bases<I3FrameObject>, I3IntPtr >("I3Int")
        .def_readwrite("value", &I3Int::value)
        ;
    def("make_int", make_int);
}

>>> i = icetray.make_int(777)
>>> i.value
777
```

Downcasting

```
I3FrameObjectPtr make_int(int i)
{
    return I3FrameObjectPtr(new I3Int(i));
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3FrameObject, I3FrameObjectPtr>("I3FrameObject",
                                                no_init);
    class_<I3Int, bases<I3FrameObject>, I3IntPtr>("I3Int")
        .def_readwrite("value", I3Int::value);
    def("make_int", make_int);
}
```

Downcasting

```
I3FrameObjectPtr make_int(int i) // boost::shared_ptr<I3FrameObject>
{
    return I3FrameObjectPtr(new I3Int(i));
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3FrameObject, I3FrameObjectPtr>("I3FrameObject",
                                                no_init);
    class_<I3Int, bases<I3FrameObject>, I3IntPtr>("I3Int")
        .def_readwrite("value", I3Int::value);
    def("make_int", make_int);
}
```

Downcasting

```
I3FrameObjectPtr make_int(int i)
{
    return I3FrameObjectPtr(new I3Int(i));
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3FrameObject, I3FrameObjectPtr>("I3FrameObject",
                                                no_init);
    class_<I3Int, bases<I3FrameObject>, I3IntPtr>("I3Int")
        .def_readwrite("value", I3Int::value);
    def("make_int", make_int);
}
```

Downcasting

```
I3FrameObjectPtr make_int(int i)
{
    return I3FrameObjectPtr(new I3Int(i));
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3FrameObject, I3FrameObjectPtr>("I3FrameObject",
                                                no_init);
    class_<I3Int, bases<I3FrameObject>, I3IntPtr>("I3Int")
        .def_readwrite("value", I3Int::value);
    def("make_int", make_int);
}
```

Downcasting

```
I3FrameObjectPtr make_int(int i)
{
    return I3FrameObjectPtr(new I3Int(i));
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3FrameObject, I3FrameObjectPtr>("I3FrameObject",
                                                no_init);
    class_<I3Int, bases<I3FrameObject>, I3IntPtr>("I3Int")
        .def_readwrite("value", I3Int::value);
    def("make_int", make_int);
}

>>> i = icetray.make_int(777)
>>>
```

Downcasting

```
I3FrameObjectPtr make_int(int i)
{
    return I3FrameObjectPtr(new I3Int(i));
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3FrameObject, I3FrameObjectPtr>("I3FrameObject",
                                                no_init);
    class_<I3Int, bases<I3FrameObject>, I3IntPtr>("I3Int")
        .def_readwrite("value", I3Int::value);
    def("make_int", make_int);
}

>>> i = icetray.make_int(777)
>>> i
<icecube.icetray.I3Int object at 0x20b1668>
>>>
```

Downcasting

```
I3FrameObjectPtr make_int(int i)
{
    return I3FrameObjectPtr(new I3Int(i));
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3FrameObject, I3FrameObjectPtr>("I3FrameObject",
                                                no_init);
    class_<I3Int, bases<I3FrameObject>, I3IntPtr>("I3Int")
        .def_readwrite("value", I3Int::value);
    def("make_int", make_int);
}

>>> i = icetray.make_int(777)
>>> i
<icecube.icetray.I3Int object at 0x20b1668>
>>>
```

Downcasting

```
I3FrameObjectPtr make_int(int i)
{
    return I3FrameObjectPtr(new I3Int(i));
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3FrameObject, I3FrameObjectPtr>("I3FrameObject",
                                                no_init);
    class_<I3Int, bases<I3FrameObject>, I3IntPtr>("I3Int")
        .def_readwrite("value", I3Int::value);
    def("make_int", make_int);
}

>>> i = icetray.make_int(777)
>>> i
<icecube.icetray.I3Int object at 0x20b1668>
>>> i.value
777
```

Dealing with overloads

```
class I3Frame {  
public:  
    void Put(const std::string& key,  
             I3FrameObjectConstPtr datum,  
             const I3Frame::Stream& stream);  
    void Put(const std::string& key,  
             I3FrameObjectConstPtr datum);  
};
```

```
BOOST_PYTHON_MODULE(icetray)  
{  
    class_<I3Frame, I3FramePtr>("I3Frame")  
        .def("Put", &I3Frame::Put);  
}
```

Dealing with overloads

```
class I3Frame {  
public:  
    void Put(const std::string& key,  
              I3FrameObjectConstPtr datum,  
              const I3Frame::Stream& stream);  
    void Put(const std::string& key,  
              I3FrameObjectConstPtr datum);  
};
```

```
BOOST_PYTHON_MODULE(icetray)  
{  
    class_<I3Frame, I3FramePtr>("I3Frame")  
        .def("Put", &I3Frame::Put);  
}
```

Dealing with overloads

```
class I3Frame {  
public:  
    void Put(const std::string& key,  
              I3FrameObjectConstPtr datum,  
              const I3Frame::Stream& stream);  
    void Put(const std::string& key,  
              I3FrameObjectConstPtr datum);  
};
```

```
error: no matching function for call to  
'boost::python::class_<I3Frame,  
                                         boost::shared_ptr<I3Frame>,>  
                                         boost::python::detail::not_specified,  
                                         boost::python::detail::not_specified>  
::def(const char [4],  
      <unresolved overloaded function type>)'
```

Dealing with overloads

```
class I3Frame {
public:
    void Put(const std::string& key,
              I3FrameObjectConstPtr datum,
              const I3Frame::Stream& stream);
    void Put(const std::string& key,
              I3FrameObjectConstPtr datum);
};

BOOST_PYTHON_MODULE(icetray)
{
    typedef void(I3Frame::*put_t)(const std::string&,
                                I3FrameObjectConstPtr);

    class_<I3Frame, I3FramePtr>("I3Frame")
        .def("Put", (put_t)&I3Frame::Put)
        ;
}
```

Dealing with overloads

```
class I3Frame {  
public:  
    void Put(const std::string& key,  
             I3FrameObjectConstPtr datum,  
             const I3Frame::Stream& stream);  
    void Put(const std::string& key,  
             I3FrameObjectConstPtr datum);  
};  
  
BOOST_PYTHON_MODULE(icetray)  
{  
    typedef void(I3Frame::*put_t)(const std::string&, I3FrameObjectConstPtr);  
  
    class_<I3Frame, I3FramePtr>("I3Frame")  
        .def("Put", (put_t)&I3Frame::Put)  
        ;  
}
```

Dealing with overloads

```
class I3Frame {  
public:  
    void Put(const std::string& key,  
             I3FrameObjectConstPtr datum,  
             const I3Frame::Stream& stream);  
    void Put(const std::string& key,  
             I3FrameObjectConstPtr datum);  
};  
  
BOOST_PYTHON_MODULE(icetray)  
{  
    typedef void(I3Frame::*put_t)(const std::string&,  
                           I3FrameObjectConstPtr);  
  
    class_<I3Frame, I3FramePtr>("I3Frame")  
        .def("Put", (put_t)&I3Frame::Put)  
        ;  
}
```

Dealing with overloads

```
class I3Frame {
public:
    void Put(const std::string& key,
              I3FrameObjectConstPtr datum,
              const I3Frame::Stream& stream);
    void Put(const std::string& key,
              I3FrameObjectConstPtr datum);
};

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Frame, I3FramePtr>("I3Frame")
        .def("Put",
             (void(I3Frame::*)(const std::string&,
                               I3FrameObjectConstPtr))
             &I3Frame::Put)
    ;
}
```

Dealing with overloads

```
class I3Frame {  
public:  
    void Put(const std::string& key,  
             I3FrameObjectConstPtr datum,  
             const I3Frame::Stream& stream);  
    void Put(const std::string& key,  
             I3FrameObjectConstPtr datum);  
};  
  
BOOST_PYTHON_MODULE(icetray)  
{  
    class_<I3Frame, I3FramePtr>("I3Frame")  
        .def("Put",  
              (void(I3Frame::*)(const std::string&,  
                           I3FrameObjectConstPtr))  
              &I3Frame::Put)  
        ;  
}
```

I3Frame::Put in action

```
I3Frame frame;  
  
>>> frame = I3Frame()
```

I3Frame::Put in action

```
I3Frame frame;
I3IntPtr thing_p(new I3Int(2012));

>>> frame = I3Frame()
>>> thing = I3Int(2012)
```

I3Frame::Put in action

```
I3Frame frame;  
I3IntPtr thing_p(new I3Int(2012));  
frame.Put("where", thing_p);  
  
>>> frame = I3Frame()  
>>> thing = I3Int(2012)  
>>> frame.Put('where', thing)
```

I3Frame::Get

```
template <typename T>
T
I3Frame::Get(const std::string& name,
             enable_if<is_shared_ptr_to_const<T> >::type* = 0)
{
    I3FrameObjectConstPtr thing = // get it ...
    return dynamic_pointer_cast<typename T::value_type>(thing);
}

I3Frame frame;
I3IntConstPtr int_p = frame.Get<I3IntConstPtr>("where");
const I3Int& int_ref = frame.Get<I3Int>("where");
```

I3Frame::Get

```
template <typename T>
T
I3Frame::Get(const std::string& name,
               enable_if<is_shared_ptr_to_const<T> >::type* = 0)
{
    I3FrameObjectConstPtr thing = // get it ...
    return dynamic_pointer_cast<typename T::value_type>(thing);
}

I3Frame frame;
I3IntConstPtr int_p = frame.Get<I3IntConstPtr>("where");
const I3Int& int_ref = frame.Get<I3Int>("where");
```

I3Frame::Get

```
template <typename T>
I3Frame::Get(const std::string& name,
             enable_if<is_shared_ptr_to_const<T> >::type* = 0)
{
    I3FrameObjectConstPtr thing = // get it ...
    return dynamic_pointer_cast<typename T::value_type>(thing);
}

I3Frame frame;
I3IntConstPtr int_p = frame.Get<I3IntConstPtr>("where");
const I3Int& int_ref = frame.Get<I3Int>("where");
```

I3Frame::Get

```
template <typename T>
T
I3Frame::Get(const std::string& name,
             enable_if<is_shared_ptr_to_const<T> >::type* = 0)
{
    I3FrameObjectConstPtr thing = // get it ...
    return dynamic_pointer_cast<typename T::value_type>(thing);
}

I3Frame frame;
I3IntConstPtr int_p = frame.Get<I3IntConstPtr>("where");
const I3Int& int_ref = frame.Get<I3Int>("where");
```

I3Frame::Get

```
template <typename T>
T
I3Frame::Get(const std::string& name,
             enable_if<is_shared_ptr_to_const<T> >::type* = 0)
{
    I3FrameObjectConstPtr thing = // get it ...
    return dynamic_pointer_cast<typename T::value_type>(thing);
}

I3Frame frame;
I3IntConstPtr int_p = frame.Get<I3IntConstPtr>("where");
const I3Int& int_ref = frame.Get<I3Int>("where");
```

I3Frame::Get

```
template <typename T>
T
I3Frame::Get(const std::string& name,
             enable_if<is_shared_ptr_to_const<T> >::type* = 0)
{
    I3FrameObjectConstPtr thing = // get it ...
    return dynamic_pointer_cast<typename T::value_type>(thing);
}

I3Frame frame;
I3IntConstPtr int_p = frame.Get<I3IntConstPtr>("where");
const I3Int& int_ref = frame.Get<I3Int>("where");
```

I3Frame::Get

```
template <typename T>
T
I3Frame::Get(const std::string& name,
             enable_if<is_shared_ptr_to_const<T> >::type* = 0)
{
    I3FrameObjectConstPtr thing = // get it ...
    return dynamic_pointer_cast<typename T::value_type>(thing);
}

I3Frame frame;
I3IntConstPtr int_p = frame.Get<I3IntConstPtr>("where");
const I3Int& int_ref = frame.Get<I3Int>("where");
```

Giving a class more member functions

```
I3FrameObjectPtr frame_get(const I3Frame* f,
                            const std::string& where)
{
    return f->Get<I3FrameObjectConstPtr>("where");
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Frame, I3FramePtr>("I3Frame")
        .def("Get", frame_get)
        ;
}

>>> frame = I3Frame()
>>> frame.Put("where", I3Int(777))
>>> i = frame.Get("where")
>>> i.value
777
```

Giving a class more member functions

```
I3FrameObjectPtr frame_get(const I3Frame* f,
                           const std::string& where)
{
    return f->Get<I3FrameObjectConstPtr>("where");
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Frame, I3FramePtr>("I3Frame")
        .def("Get", frame_get)
        ;
}

>>> frame = I3Frame()
>>> frame.Put("where", I3Int(777))
>>> i = frame.Get("where")
>>> i.value
777
```

Giving a class more member functions

```
I3FrameObjectPtr frame_get(const I3Frame* f,  
                           const std::string& where)  
{  
    return f->Get<I3FrameObjectConstPtr>("where");  
}  
  
BOOST_PYTHON_MODULE(icetray)  
{  
    class_<I3Frame, I3FramePtr>("I3Frame")  
        .def("Get", frame_get)  
        ;  
}  
  
>>> frame = I3Frame()  
>>> frame.Put("where", I3Int(777))  
>>> i = frame.Get("where")  
>>> i.value  
777
```

Giving a class more member functions

```
I3FrameObjectPtr frame_get(const I3Frame* f,
                           const std::string& where)
{
    return f->Get<I3FrameObjectConstPtr>("where");
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Frame, I3FramePtr>("I3Frame")
        .def("Get", frame_get)
        ;
}

>>> frame = I3Frame()
>>> frame.Put("where", I3Int(777))
>>> i = frame.Get("where")
>>> i.value
777
```

Giving a class more member functions

```
I3FrameObjectPtr frame_get(const I3Frame* f,
                           const std::string& where)
{ // some const_casting not shown
    return f->Get<I3FrameObjectConstPtr>("where");
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Frame, I3FramePtr>("I3Frame")
        .def("Get", frame_get)
        ;
}

>>> frame = I3Frame()
>>> frame.Put("where", I3Int(777))
>>> i = frame.Get("where")
>>> i.value
777
```

Giving a class more member functions

```
I3FrameObjectPtr frame_get(const I3Frame* f,
                           const std::string& where)
{
    return f->Get<I3FrameObjectConstPtr>("where");
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Frame, I3FramePtr>("I3Frame")
        .def("Get", frame_get)
        ;
}

>>> frame = I3Frame()
>>> frame.Put("where", I3Int(777))
>>> i = frame.Get("where")
>>> i.value
777
```

Making things look 'pythonic'

d['foo'] d['foo'] = 'bar' 'foo' in d del d['foo'] len(d)	d.__getitem__('foo') d.__setitem__('foo', 'bar') d.__contains__('foo') d.__delitem__('foo') d.__len__()
--	---

Making things look 'pythonic'

```
BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Frame, I3FramePtr>("I3Frame")
        .def("__setitem__", (put_t)&I3Frame::Put)
        .def("__getitem__", frame_get)
        .def("__len__", &I3Frame::size)
        ;
}
}

>>> frame = icetray.I3Frame()
>>> frame['where'] = I3Int(777)
>>> i = frame['where']
>>> i.value
777
>>> len(frame)
1
```

Making things look 'pythonic'

```
BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Frame, I3FramePtr>("I3Frame")
        .def("__setitem__", (put_t)&I3Frame::Put)
        .def("__getitem__", frame_get)
        .def("__len__", &I3Frame::size)
        ;
}
}

>>> frame = icetray.I3Frame()
>>> frame['where'] = I3Int(777)
>>> i = frame['where']
>>> i.value
777
>>> len(frame)
1
```

Making things look 'pythonic'

```
BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Frame, I3FramePtr>("I3Frame")
        .def("__setitem__", (put_t)&I3Frame::Put)
.def("__getitem__", frame_get)
        .def("__len__", &I3Frame::size)
        ;
}
}

>>> frame = icetray.I3Frame()
>>> frame['where'] = I3Int(777)
>>> i = frame['where']
>>> i.value
777
>>> len(frame)
1
```

Making things look 'pythonic'

```
BOOST_PYTHON_MODULE(icetray)
{
    class_<I3Frame, I3FramePtr>("I3Frame")
        .def("__setitem__", (put_t)&I3Frame::Put)
        .def("__getitem__", frame_get)
        .def("__len__", &I3Frame::size)
    ;
}
>>> frame = icetray.I3Frame()
>>> frame['where'] = I3Int(777)
>>> i = frame['where']
>>> i.value
777
>>> len(frame)
1
```

An I3File class

```
class I3File
{
    I3File(const std::string&);
    bool more() const;
    I3FramePtr next();
};

I3File f("/path/to/data.i3");
std::vector<int> v;
while (f.more())
{
    I3FramePtr frame = f.next();
    v.push_back(frame.Get<I3Int>("where")->value);
}
```

An I3File class

```
class I3File
{
    I3File(const std::string&);
    bool more() const;
    I3FramePtr next();
};

I3File f("/path/to/data.i3");
std::vector<int> v;
while (f.more())
{
    I3FramePtr frame = f.next();
    v.push_back(frame.Get<I3Int>("where")->value);
}
```

An I3File class

```
class I3File
{
    I3File(const std::string&);
    bool more() const;
    I3FramePtr next();
};

I3File f("/path/to/data.i3");
std::vector<int> v;
while (f.more())
{
    I3FramePtr frame = f.next();
    v.push_back(frame.Get<I3Int>("where")->value);
}
```

An I3File class

```
class I3File
{
    I3File(const std::string&);
    bool more() const;
    I3FramePtr next();
};

f = I3File('/path/to/data.i3')
v = []
while f.more():
    frame = f.next()
    v.append(frame["where"].value)
```

Python iterators

```
>>> a = [1,2]
>>> for x in a:
...     print x
...
1
2
>>> [x for x in a]
[1, 2]
>>> [x*3 for x in a]
[3, 6]
>>> iter = a.__iter__()
>>> iter.next()
1
>>> iter.next()
2
>>> iter.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Python iterators

```
>>> a = [1, 2]
>>> for x in a:
...     print x
...
1
2
>>> [x for x in a]
[1, 2]
>>> [x*3 for x in a]
[3, 6]
>>> iter = a.__iter__()
>>> iter.next()
1
>>> iter.next()
2
>>> iter.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Python iterators

```
>>> a = [1,2]
>>> for x in a:
...     print x
...
1
2
>>> [x for x in a]
[1, 2]
>>> [x*3 for x in a]
[3, 6]
>>> iter = a.__iter__()
>>> iter.next()
1
>>> iter.next()
2
>>> iter.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Python iterators

```
>>> a = [1,2]
>>> for x in a:
...     print x
...
1
2
>>> [x for x in a]
[1, 2]
>>> [x*3 for x in a]
[3, 6]
>>> iter = a.__iter__()
>>> iter.next()
1
>>> iter.next()
2
>>> iter.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Python iterators

```
>>> a = [1,2]
>>> for x in a:
...     print x
...
1
2
>>> [x for x in a]
[1, 2]
>>> [x*3 for x in a]
[3, 6]
>>> iter = a.__iter__()
>>> iter.next()
1
>>> iter.next()
2
>>> iter.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Python iterators

```
>>> a = [1,2]
>>> for x in a:
...     print x
...
1
2
>>> [x for x in a]
[1, 2]
>>> [x*3 for x in a]
[3, 6]
>>> iter = a.__iter__()
>>> iter.next()
1
>>> iter.next()
2
>>> iter.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Python iterators

```
>>> a = [1,2]
>>> for x in a:
...     print x
...
1
2
>>> [x for x in a]
[1, 2]
>>> [x*3 for x in a]
[3, 6]
>>> iter = a.__iter__()
>>> iter.next()
1
>>> iter.next()
2
>>> iter.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Python iterators

```
>>> a = [1,2]
>>> for x in a:
...     print x
...
1
2
>>> [x for x in a]
[1, 2]
>>> [x*3 for x in a]
[3, 6]
>>> iter = a.__iter__()
>>> iter.next()
1
>>> iter.next()
2
>>> iter.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Constructing an I3File iterator

```
struct I3File_iter
{
    boost::shared_ptr<I3File> i3file;

    I3File_iter(boost::shared_ptr<I3File> f) : i3file(f)
    { }

    I3FramePtr next()
    {
        if (i3file->more())
            return i3file->next();
        else
        {
            PyErr_SetObject(PyExc_StopIteration, Py_None);
            boost::python::throw_error_already_set();
        }
    }
}
```

Constructing an I3File iterator

```
struct I3File_iter
{
    boost::shared_ptr<I3File> i3file;

    I3File_iter(boost::shared_ptr<I3File> f) : i3file(f)
    { }

    I3FramePtr next()
    {
        if (i3file->more())
            return i3file->next();
        else
        {
            PyErr_SetObject(PyExc_StopIteration, Py_None);
            boost::python::throw_error_already_set();
        }
    }
}
```

Constructing an I3File iterator

```
struct I3File_iter
{
    boost::shared_ptr<I3File> i3file;

    I3File_iter(boost::shared_ptr<I3File> f) : i3file(f)
    { }

    I3FramePtr next()
    {
        if (i3file->more())
            return i3file->next();
        else
        {
            PyErr_SetObject(PyExc_StopIteration, Py_None);
            boost::python::throw_error_already_set();
        }
    }
}
```

Constructing an I3File iterator

```
struct I3File_iter
{
    boost::shared_ptr<I3File> i3file;

I3File_iter(boost::shared_ptr<I3File> f) : i3file(f)
{ }

I3FramePtr next()
{
    if (i3file->more())
        return i3file->next();
    else
    {
        PyErr_SetObject(PyExc_StopIteration, Py_None);
        boost::python::throw_error_already_set();
    }
}
```

Constructing an I3File iterator

```
struct I3File_iter
{
    boost::shared_ptr<I3File> i3file;

    I3File_iter(boost::shared_ptr<I3File> f) : i3file(f)
    { }

I3FramePtr next()
{
    if (i3file->more())
        return i3file->next();
    else
    {
        PyErr_SetObject(PyExc_StopIteration, Py_None);
        boost::python::throw_error_already_set();
    }
}
```

Constructing an I3File iterator

```
struct I3File_iter
{
    boost::shared_ptr<I3File> i3file;

    I3File_iter(boost::shared_ptr<I3File> f) : i3file(f)
    { }

    I3FramePtr next()
    {
        if (i3file->more())
            return i3file->next();
        else
        {
            PyErr_SetObject(PyExc_StopIteration, Py_None);
            boost::python::throw_error_already_set();
        }
    }
}
```

Constructing an I3File iterator

```
struct I3File_iter
{
    boost::shared_ptr<I3File> i3file;

    I3File_iter(boost::shared_ptr<I3File> f) : i3file(f)
    { }

    I3FramePtr next()
    {
        if (i3file->more())
            return i3file->next();
        else
        {
            PyErr_SetObject (PyExc_StopIteration, Py_None);
            boost::python::throw_error_already_set();
        }
    }
}
```

Constructing an I3File iterator

```
struct I3File_iter
{
    boost::shared_ptr<I3File> i3file;

    I3File_iter(boost::shared_ptr<I3File> f) : i3file(f)
    { }

    I3FramePtr next()
    {
        if (i3file->more())
            return i3file->next();
        else
        {
            PyErr_SetObject(PyExc_StopIteration, Py_None);
boost::python::throw_error_already_set();
        }
    }
}
```

Python iterators 2

```
I3File_iter make_iter(I3FilePtr f)
{
    I3FilePtr newfile(new I3File(*f));
    newfile->rewind();
    return I3File_iter(newfile);
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3File, I3FilePtr>("I3File")
        .def(init<const std::string&>())
        .def("__iter__", make_iter)
        ;
}
```

Python iterators 2

```
I3File_iter make_iter(I3FilePtr f)
{
    I3FilePtr newfile(new I3File(*f));
    newfile->rewind();
    return I3File_iter(newfile);
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3File, I3FilePtr>("I3File")
        .def(init<const std::string&>())
        .def("__iter__", make_iter)
    ;
}
```

Python iterators 2

```
I3File_iter make_iter(I3FilePtr f)
{
    I3FilePtr newfile(new I3File(*f));
    newfile->rewind();
    return I3File_iter(newfile);
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3File, I3FilePtr>("I3File")
        .def(init<const std::string&>())
        .def("__iter__", make_iter)
    ;
}
```

Python iterators 2

```
I3File_iter make_iter(I3FilePtr f)
{
    I3FilePtr newfile(new I3File(*f));
    newfile->rewind();
    return I3File_iter(newfile);
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3File, I3FilePtr>("I3File")
        .def(init<const std::string&>())
        .def("__iter__", make_iter)
        ;
}
```

Python iterators 2

```
I3File_iter make_iter(I3FilePtr f)
{
    I3FilePtr newfile(new I3File(*f));
    newfile->rewind();
    return I3File_iter(newfile);
}

BOOST_PYTHON_MODULE(icetray)
{
    class_<I3File, I3FilePtr>("I3File")
        .def(init<const std::string&>())
        .def("__iter__", make_iter)
        ;
}
```

Python iterators 2

```
>>> for frame in I3File("/some/data.i3"):  
...     print frame["where"].value  
...  
1  
2  
3  
>>> [frame["where"].value  
...     for frame in I3File("/some/data.i3")]  
[1, 2, 3]
```

Python iterators 2

```
>>> for frame in I3File("/some/data.i3"):  
...     print frame["where"].value  
...  
1  
2  
3  
>>> [frame["where"].value  
...     for frame in I3File("/some/data.i3")  
...     if "where" in frame]  
[1, 2, 3]
```

Python iterators 2

```
>>> for frame in I3File("/some/data.i3"):  
...     print frame["where"].value  
...  
1  
2  
3  
>>> [frame["where"].value  
...     for frame in I3File("/some/data.i3")  
...     if len(frame["recohits"]) > 80]  
[1, 2, 3]
```

vector containers

```
class_<std::vector<double> >("vector_double")
    .def(vector_indexing_suite<std::vector<double> >())
;

>>> vd = icetray.vector_double()
>>> vd.append(3)
>>> vd.append(4)
>>> vd[0]
3.0
>>> [x*17 for x in vd]
[51.0, 68.0]
>>> len(vd)
2
```

associative containers

```
class_<std::map<string, double> >("map_string_double")
    .def(map_indexing_suite<std::map<string, double> >())
;

>>> mii = icetray.map_int_int()
>>> msd[3]=4
>>> msd[5]=6
>>> for entry in msd:
...     print entry.key(), "=>", entry.data()
...
3 => 4
5 => 6
```

I3Module redux

```
class I3Module
{
    virtual void Physics(I3FramePtr frame) /* boring default*/
};

class MyFilterModule : public I3Module
{
    void Physics(I3FramePtr frame)
    {
        if (frame.Has("something"))
            PushFrame(frame);
    }
};

I3_MODULE(MyFilterModule);
```

I3Module redux

```
class I3Module
{
    virtual void Physics(I3FramePtr frame) /* boring default*/
};

class MyFilterModule : public I3Module
{
    void Physics(I3FramePtr frame)
    {
        if (frame.Has("something"))
            PushFrame(frame);
    }
};

I3_MODULE(MyFilterModule);
```

I3Module redux

```
class I3Module
{
    virtual void Physics(I3FramePtr frame) /* boring default*/
};

class MyFilterModule : public I3Module
{
    void Physics(I3FramePtr frame)
    {
        if (frame.Has("something"))
            PushFrame(frame);
    }
};

I3_MODULE (MyFilterModule);
```

I3Module redux

```
class I3Module
{
    virtual void Physics(I3FramePtr frame) /* boring default*/
};

class MyFilterModule : public I3Module
{
    void Physics(I3FramePtr frame)
    {
        if (frame.Has("something"))
            PushFrame(frame);
    }
};

I3_MODULE(MyFilterModule);
```

I3Module redux

```
class I3Module
{
    virtual void Physics(I3FramePtr frame) /* boring default*/
};

class MyFilterModule : public I3Module
{
    void Physics(I3FramePtr frame)
    {
        if (frame.Has("something"))
            PushFrame(frame);
    }
};

I3_MODULE (MyFilterModule);
```

Using the module

```
from icecube import icetray

tray = icetray.I3Tray()

tray.AddModule("I3Reader", "reader",
               Filename = "input_data.i3")

tray.AddModule("MyFilterModule", "m")

tray.AddModule("I3Writer", "writer",
               Filename = "outdata.i3")

tray.Execute()
```

Using the module

```
from icecube import icetray

tray = icetray.I3Tray()

tray.AddModule("I3Reader", "reader",
               Filename = "input_data.i3")

tray.AddModule("MyFilterModule", "m")

tray.AddModule("I3Writer", "writer",
               Filename = "outdata.i3")

tray.Execute()
```

Inheriting from a C++ base class... in python (wrapper side)

```
class PythonModule : I3Module, wrapper<I3Module>
{
    void Physics(I3FramePtr frame)
    {
        if (bp::override ds = this->get_override("Physics"))
            ds(frame);
        else
            I3Module::Physics(frame);
    }
};

class_<PythonModule, PythonModulePtr> ("I3Module")
    .def("Physics", &PythonModule::Physics)
    .def("PushFrame", &PythonModule::PushFrame)
;
```

Inheriting from a C++ base class... in python (wrapper side)

```
class PythonModule : I3Module, wrapper<I3Module>
{
    void Physics(I3FramePtr frame)
    {
        if (bp::override ds = this->get_override("Physics"))
            ds(frame);
        else
            I3Module::Physics(frame);
    }
};

class_<PythonModule, PythonModulePtr> ("I3Module")
    .def("Physics", &PythonModule::Physics)
    .def("PushFrame", &PythonModule::PushFrame)
;
```

Inheriting from a C++ base class... in python (wrapper side)

```
class PythonModule : I3Module, wrapper<I3Module>
{
    void Physics(I3FramePtr frame)
    {
        if (bp::override ds = this->get_override("Physics"))
            ds(frame);
        else
            I3Module::Physics(frame);
    }
};

class_<PythonModule, PythonModulePtr> ("I3Module")
    .def("Physics", &PythonModule::Physics)
    .def("PushFrame", &PythonModule::PushFrame)
;
```

Inheriting from a C++ base class... in python (wrapper side)

```
class PythonModule : I3Module, wrapper<I3Module>
{
    void Physics(I3FramePtr frame)
    {
        if (bp::override ds = this->get_override("Physics"))
            ds(frame);
        else
            I3Module::Physics(frame);
    }
};

class_<PythonModule, PythonModulePtr> ("I3Module")
    .def("Physics", &PythonModule::Physics)
    .def("PushFrame", &PythonModule::PushFrame)
;
```

Inheriting from a C++ base class... in python (wrapper side)

```
class PythonModule : I3Module, wrapper<I3Module>
{
    void Physics(I3FramePtr frame)
    {
        if (bp::override ds = this->get_override("Physics"))
            ds(frame);
        else
            I3Module::Physics(frame);
    }
};

class_<PythonModule, PythonModulePtr> ("I3Module")
    .def("Physics", &PythonModule::Physics)
    .def("PushFrame", &PythonModule::PushFrame)
;
```

Inheriting from a C++ base class... in python (wrapper side)

```
class PythonModule : I3Module, wrapper<I3Module>
{
    void Physics(I3FramePtr frame)
    {
        if (bp::override ds = this->get_override("Physics"))
            ds(frame);
        else
I3Module::Physics(frame);
    }
};

class_<PythonModule, PythonModulePtr> ("I3Module")
    .def("Physics", &PythonModule::Physics)
    .def("PushFrame", &PythonModule::PushFrame)
;
```

Inheriting from a C++ base class... in python (wrapper side)

```
class PythonModule : I3Module, wrapper<I3Module>
{
    void Physics(I3FramePtr frame)
    {
        if (bp::override ds = this->get_override("Physics"))
            ds(frame);
        else
            I3Module::Physics(frame);
    }
};

class_<PythonModule, PythonModulePtr> ("I3Module")
    .def("Physics", &PythonModule::Physics)
    .def("PushFrame", &PythonModule::PushFrame)
;
```

Inheriting from a C++ base class... in python (wrapper side)

```
class PythonModule : I3Module, wrapper<I3Module>
{
    void Physics(I3FramePtr frame)
    {
        if (bp::override ds = this->get_override("Physics"))
            ds(frame);
        else
            I3Module::Physics(frame);
    }
};

class_<PythonModule, PythonModulePtr> ("I3Module")
    .def("Physics", &PythonModule::Physics)
    .def("PushFrame", &PythonModule::PushFrame)
;
```

Inheriting from a C++ base class... in python (python side)

```
from icecube.icetray import *

class MyFilterModule(I3Module):

    def __init__(self):
        I3Module.__init__(self)

    def Physics(self, frame):
        if "something" in frame:
            self.PushFrame(frame)

tray.AddModule("I3Reader", "reader",
               Filename = "input_data.i3")

tray.AddModule(MyFilterModule, "filter")

tray.AddModule("I3Writer", "writer",
               Filename = "output_data.i3")
```

Inheriting from a C++ base class... in python (python side)

```
from icecube.icetray import *

class MyFilterModule(I3Module):

    def __init__(self):
        I3Module.__init__(self)

    def Physics(self, frame):
        if "something" in frame:
            self.PushFrame(frame)

tray.AddModule("I3Reader", "reader",
               Filename = "input_data.i3")

tray.AddModule(MyFilterModule, "filter")

tray.AddModule("I3Writer", "writer",
               Filename = "output_data.i3")
```

Inheriting from a C++ base class... in python (python side)

```
from icecube.icetray import *

class MyFilterModule(I3Module):

    def __init__(self):
        I3Module.__init__(self)

    def Physics(self, frame):
        if "something" in frame:
            self.PushFrame(frame)

tray.AddModule("I3Reader", "reader",
               Filename = "input_data.i3")

tray.AddModule(MyFilterModule, "filter")

tray.AddModule("I3Writer", "writer",
               Filename = "output_data.i3")
```

Inheriting from a C++ base class... framework side

```
namespace bp = boost::python;

void I3Tray::AddModule(bp::object type_object,
                      const std::string& instancename)
{
    assert(PyType_Check(type_object.ptr()));
    bp::object instance = type_object();
    I3ModulePtr mod_p = bp::extract<I3ModulePtr>(instance);

    // next:
    // 1. put the mod_p with all the other modules and remain
    //     blissfully ignorant of the fact that the calls go
    //     through to python
    // 2. ???
    // 3. profit
}
```

Inheriting from a C++ base class... framework side

```
namespace bp = boost::python;

void I3Tray::AddModule(bp::object type_object,
                      const std::string& instancename)
{
    assert(PyType_Check(type_object.ptr()));
    bp::object instance = type_object();
    I3ModulePtr mod_p = bp::extract<I3ModulePtr>(instance);

    // next:
    // 1. put the mod_p with all the other modules and remain
    //     blissfully ignorant of the fact that the calls go
    //     through to python
    // 2. ???
    // 3. profit
}
```

Inheriting from a C++ base class... framework side

```
namespace bp = boost::python;

void I3Tray::AddModule(bp::object type_object,
                      const std::string& instancename)
{
    assert(PyType_Check(type_object.ptr()));
bp::object instance = type_object();
    I3ModulePtr mod_p = bp::extract<I3ModulePtr>(instance);

    // next:
    // 1. put the mod_p with all the other modules and remain
    //     blissfully ignorant of the fact that the calls go
    //     through to python
    // 2. ???
    // 3. profit
}
```

Inheriting from a C++ base class... framework side

```
namespace bp = boost::python;

void I3Tray::AddModule(bp::object type_object,
                      const std::string& instancename)
{
    assert(PyType_Check(type_object.ptr()));
    bp::object instance = type_object();
I3ModulePtr mod_p = bp::extract<I3ModulePtr>(instance);

    // next:
    // 1. put the mod_p with all the other modules and remain
    //     blissfully ignorant of the fact that the calls go
    //     through to python
    // 2. ???
    // 3. profit
}
```

Functions as modules

```
from icecube.icetray import *

class MyFilterModule(I3Module):

    def __init__(self):
        I3Module.__init__(self)

    def Physics(self, frame):
        if "something" in frame:
            self.PushFrame(frame)

tray.AddModule("I3Reader", "reader",
               Filename = "input_data.i3")

tray.AddModule(MyFilterModule, "filter")

tray.AddModule("I3Writer", "writer",
               Filename = "output_data.i3")
```

Functions as modules

```
from icecube.icetray import *

def my_predicate(frame):
    return "something" in frame

tray.AddModule("I3Reader", "reader",
               Filename = "input_data.i3")

tray.AddModule(my_predicate, "filter")

tray.AddModule("I3Writer", "writer",
               Filename = "output_data.i3")
```

Functions as modules

```
from icecube.icetray import *

tray.AddModule("I3Reader", "reader",
               Filename = "input_data.i3")

tray.AddModule(lambda frame: "something" in frame, "filter")

tray.AddModule("I3Writer", "writer",
               Filename = "output_data.i3")
```

An module that thunks to a python function

```
class PythonFunction : public I3Module
{
    bp::object fn;

    void Physics(I3FramePtr frame)
    {
        bp::object return_value = fn(frame);
        if (return_value.ptr() == Py_None)
        {
            PushFrame(frame);
            return;
        }
        bool flag = bp::extract<bool>(return_value);
        if (flag)
            PushFrame(frame);
        return;
    }
};
```

An module that thunks to a python function

```
class PythonFunction : public I3Module
{
    bp::object fn;

    void Physics(I3FramePtr frame)
    {
        bp::object return_value = fn(frame);
        if (return_value.ptr() == Py_None)
        {
            PushFrame(frame);
            return;
        }
        bool flag = bp::extract<bool>(return_value);
        if (flag)
            PushFrame(frame);
        return;
    }
};
```

An module that thunks to a python function

```
class PythonFunction : public I3Module
{
    bp::object fn;

    void Physics(I3FramePtr frame)
    {
        bp::object return_value = fn(frame);
        if (return_value.ptr() == Py_None)
        {
            PushFrame(frame);
            return;
        }
        bool flag = bp::extract<bool>(return_value);
        if (flag)
            PushFrame(frame);
        return;
    }
};
```

An module that thunks to a python function

```
class PythonFunction : public I3Module
{
    bp::object fn;

void Physics(I3FramePtr frame)
{
    bp::object return_value = fn(frame);
    if (return_value.ptr() == Py_None)
    {
        PushFrame(frame);
        return;
    }
    bool flag = bp::extract<bool>(return_value);
    if (flag)
        PushFrame(frame);
    return;
}
};
```

An module that thunks to a python function

```
class PythonFunction : public I3Module
{
    bp::object fn;

    void Physics(I3FramePtr frame)
    {
        bp::object return_value = fn(frame);
        if (return_value.ptr() == Py_None)
        {
            PushFrame(frame);
            return;
        }
        bool flag = bp::extract<bool>(return_value);
        if (flag)
            PushFrame(frame);
        return;
    }
};
```

An module that thunks to a python function

```
class PythonFunction : public I3Module
{
    bp::object fn;

    void Physics(I3FramePtr frame)
    {
        bp::object return_value = fn(frame);
        if (return_value.ptr() == Py_None)
        {
            PushFrame(frame);
            return;
        }
        bool flag = bp::extract<bool>(return_value);
        if (flag)
            PushFrame(frame);
        return;
    }
};
```

An module that thunks to a python function

```
class PythonFunction : public I3Module
{
    bp::object fn;

    void Physics(I3FramePtr frame)
    {
        bp::object return_value = fn(frame);
        if (return_value.ptr() == Py_None)
        {
            PushFrame(frame);
            return;
        }
        bool flag = bp::extract<bool>(return_value);
        if (flag)
            PushFrame(frame);
        return;
    }
};
```

An module that thunks to a python function

```
class PythonFunction : public I3Module
{
    bp::object fn;

    void Physics(I3FramePtr frame)
    {
        bp::object return_value = fn(frame);
        if (return_value.ptr() == Py_None)
        {
            PushFrame(frame);
            return;
        }
        bool flag = bp::extract<bool>(return_value);
if (flag)
    PushFrame(frame);
    return;
}
};
```

Emulating an overload on the C++ side

```
namespace bp = boost::python;

void I3Tray::AddModule(bp::object obj,
                      const std::string& instancename)
{
    if(PyType_Check(obj.ptr())) {
        bp::object instance = obj();
        I3ModulePtr mod_p = bp::extract<I3ModulePtr>(instance)
            // put mod_p alongside the other modules
    } else if (PyFunction_Check(obj.ptr())) {
        PythonFunctionPtr func_module(new PythonFunction(...))
        func_module->fn = obj;
        // put func_module alongside the other modules
    }
    // etc
}
```

Packaging

```
lib/
    libexamples.so
icecube/
    __init__.py
    load_pybindings.py
    examples/
        examples.so
        OtherStuff.py
        __init__.py
```

```
#!/usr/bin/python
```

```
from icecube.examples import CppClass, PurePythonClass
```

Packaging

```
lib/
  libexamples.so          # C++ only, ignorant of python
  icecube/
    __init__.py
    load_pybindings.py
  examples/
    examples.so
    OtherStuff.py
    __init__.py

class CppClass {
public:
  void foo();
};
```

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/              # named after the 'suite'
        __init__.py
        load_pybindings.py
    examples/
        examples.so
        OtherStuff.py
        __init__.py
```

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/
        __init__.py         # named after the 'suite'
        load_pybindings.py
        examples/
            examples.so
            OtherStuff.py
            __init__.py
```

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/
        __init__.py         # named after the 'suite'
        load_pybindings.py # marks dir as python package
        # magic here
    examples/
        examples.so
        OtherStuff.py
        __init__.py
```

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/
        __init__.py         # named after the 'suite'
        load_pybindings.py  # marks dir as python package
        examples/           # magic here
            examples.so
            OtherStuff.py
            __init__.py
```

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/
        __init__.py         # named after the 'suite'
        load_pybindings.py  # marks dir as python package
    examples/
        examples.so          # magic here
        # For each project
        examples.so          # Compiled python bindings
        OtherStuff.py
        __init__.py

BOOST_PYTHON_MODULE(examples)
{
    class_<CppClass>("CppClass")
        .def("foo", &CppClass::foo)
    ;
}
```

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/
        __init__.py         # named after the 'suite'
        load_pybindings.py  # marks dir as python package
        examples/
            examples.so      # magic here
            # For each project
            # Compiled python bindings
            OtherStuff.py     # pure python components
            __init__.py

def PurePythonClass:
    def __init__(self):
        pass

    def foo(x):
        return x ** 10

def python_function(x, y, *args, **kwargs):
    pass
```

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/
        __init__.py         # named after the 'suite'
        load_pybindings.py  # marks dir as python package
        examples/
            examples.so      # magic here
            # For each project
            OtherStuff.py    # Compiled python bindings
            __init__.py       # pure python components
                            # calls up to load_pybindings
```

if we call “**from icecube import examples**”

```
import icecube.load_pybindings
icecube.load_pybindings(__name__, __path__)
```

-

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/
        __init__.py         # named after the 'suite'
        load_pybindings.py  # marks dir as python package
    examples/
        examples.so          # For each project
        OtherStuff.py        # pure python components
        __init__.py          # calls up to load_pybindings
```

if we call "from icecube import examples"

```
import icecube.load_pybindings
icecube.load_pybindings(__name__, __path__)
```

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/
        __init__.py         # named after the 'suite'
        load_pybindings.py  # marks dir as python package
        examples/
            examples.so      # For each project
            OtherStuff.py    # Compiled python bindings
            __init__.py       # pure python components
                                # calls up to load_pybindings
```

if we call "from icecube import examples"

```
import icecube.load_pybindings
icecube.load_pybindings(__name__, __path__)

"icecube.examples"
```

-

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/
        __init__.py         # named after the 'suite'
        load_pybindings.py  # marks dir as python package
    examples/
        examples.so          # For each project
        OtherStuff.py        # pure python components
        __init__.py          # calls up to load_pybindings
```

if we call "from icecube import examples"

```
import icecube.load_pybindings
icecube.load_pybindings(__name__, __path__)
"/path/to/lib/examples"
```

-

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/
        __init__.py         # named after the 'suite'
        load_pybindings.py # marks dir as python package
    examples/
        examples.so          # For each project
        OtherStuff.py        # pure python components
        __init__.py          # calls up to load_pybindings

def load_pybindings(name, path):

    import imp, sys
    cpp_module = imp.load_dynamic(name, path[0] + ".so")
    this_module = sys.modules[name]

    for (k,v) in cpp_module.__dict__.items():
        if not k.startswith("_"):
            this_module.__dict__[k] = v
```

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/
        __init__.py         # named after the 'suite'
        load_pybindings.py # marks dir as python package
    examples/
        examples.so          # For each project
        OtherStuff.py        # pure python components
        __init__.py          # calls up to load_pybindings

def load_pybindings(name, path):

    import imp, sys
    cpp_module = imp.load_dynamic(name, path[0] + ".so")
    this_module = sys.modules[name]

    for (k,v) in cpp_module.__dict__.items():
        if not k.startswith("_"):
            this_module.__dict__[k] = v
```

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/
        __init__.py         # named after the 'suite'
        load_pybindings.py # marks dir as python package
    examples/
        examples.so          # For each project
        OtherStuff.py        # pure python components
        __init__.py          # calls up to load_pybindings

def load_pybindings(name, path):
    import imp, sys
    cpp_module = imp.load_dynamic(name, path[0] + ".so")
    this_module = sys.modules[name]

    for (k,v) in cpp_module.__dict__.items():
        if not k.startswith("_"):
            this_module.__dict__[k] = v
```

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/
        __init__.py         # marks dir as python package
        load_pybindings.py # magic here
    examples/
        examples.so         # For each project
        OtherStuff.py       # pure python components
        __init__.py         # calls up to load_pybindings

def load_pybindings(name, path):
    import imp, sys
    cpp_module = imp.load_dynamic(name, path[0] + ".so")
this_module = sys.modules[name]

    for (k,v) in cpp_module.__dict__.items():
        if not k.startswith("_"):
            this_module.__dict__[k] = v
```

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/
        __init__.py         # named after the 'suite'
        load_pybindings.py # marks dir as python package
    examples/
        examples.so          # For each project
        OtherStuff.py        # pure python components
        __init__.py          # calls up to load_pybindings

def load_pybindings(name, path):

    import imp, sys
    cpp_module = imp.load_dynamic(name, path[0] + ".so")
    this_module = sys.modules[name]
    # "CppClass"
    for (k,v) in cpp_module.__dict__.items():
        if not k.startswith("_"):
            this_module.__dict__[k] = v
```

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/
        __init__.py         # marks dir as python package
        load_pybindings.py # magic here
    examples/
        examples.so         # For each project
        OtherStuff.py       # pure python components
        __init__.py         # calls up to load_pybindings

def load_pybindings(name, path):

    import imp, sys
    cpp_module = imp.load_dynamic(name, path[0] + ".so")
    this_module = sys.modules[name]
    # the actual class object
    for (k,v) in cpp_module.__dict__.items():
        if not k.startswith("_"):
            this_module.__dict__[k] = v
```

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/
        __init__.py         # named after the 'suite'
        load_pybindings.py # marks dir as python package
    examples/
        examples.so          # For each project
        OtherStuff.py        # pure python components
        __init__.py          # calls up to load_pybindings

def load_pybindings(name, path):

    import imp, sys
    cpp_module = imp.load_dynamic(name, path[0] + ".so")
    this_module = sys.modules[name]

    for (k,v) in cpp_module.__dict__.items():
        if not k.startswith("_"):
            this_module.__dict__[k] = v
```

Packaging

```
lib/
    libexamples.so          # C++ only, ignorant of python
    icecube/
        __init__.py         # marks dir as python package
        load_pybindings.py # magic here
    examples/
        examples.so         # For each project
        OtherStuff.py       # pure python components
        __init__.py         # calls up to load_pybindings

def load_pybindings(name, path):
    import imp, sys
    cpp_module = imp.load_dynamic(name, path[0] + ".so")
    this_module = sys.modules[name]

    for (k,v) in cpp_module.__dict__.items():
        if not k.startswith("_"):
            this_module.__dict__[k] = v
```

INI files are boring.

```
[server]
description = configure me
port = 26227
```

Python as the ultimate config file language

```
#  
# description - publically advertised  
# description of data  
#  
desc = "configure me"  
#  
# port - port # on which to start server  
# beware of privs on lower port #s  
#  
port = 26227
```

From the c++ side

```
namespace bp = boost::python;

int main(int argc, char** argv)
{
    Py_Initialize();
    PyEval_InitThreads();
    bp::object main = bp::import("__main__");
    bp::object interpreter = main.attr("__dict__");
    bp::exec_file(argv[1], interpreter, interpreter);

    int port = bp::extract<int>(interpreter["port"]);
    std::string desc =
        bp::extract<std::string>(interpreter["desc"]);
    std::cout << "Starting server " << desc
        << " on port " << port;
}
```

From the c++ side

```
namespace bp = boost::python;

int main(int argc, char** argv)
{
    Py_Initialize();
    PyEval_InitThreads();
    bp::object main = bp::import("__main__");
    bp::object interpreter = main.attr("__dict__");
    bp::exec_file(argv[1], interpreter, interpreter);

    int port = bp::extract<int>(interpreter["port"]);
    std::string desc =
        bp::extract<std::string>(interpreter["desc"]);
    std::cout << "Starting server " << desc
        << " on port " << port;
}
```

From the c++ side

```
namespace bp = boost::python;

int main(int argc, char** argv)
{
    Py_Initialize();
    PyEval_InitThreads();
    bp::object main = bp::import("__main__");
    bp::object interpreter = main.attr("__dict__");
    bp::exec_file(argv[1], interpreter, interpreter);

    int port = bp::extract<int>(interpreter["port"]);
    std::string desc =
        bp::extract<std::string>(interpreter["desc"]);
    std::cout << "Starting server " << desc
        << " on port " << port;
}
```

From the c++ side

```
namespace bp = boost::python;

int main(int argc, char** argv)
{
    Py_Initialize();
    PyEval_InitThreads();
    bp::object main = bp::import("__main__");
    bp::object interpreter = main.attr("__dict__");
    bp::exec_file(argv[1], interpreter, interpreter);

    int port = bp::extract<int>(interpreter["port"]);
    std::string desc =
        bp::extract<std::string>(interpreter["desc"]);
    std::cout << "Starting server " << desc
        << " on port " << port;
}
```

From the c++ side

```
namespace bp = boost::python;

int main(int argc, char** argv)
{
    Py_Initialize();
    PyEval_InitThreads();
    bp::object main = bp::import("__main__");
    bp::object interpreter = main.attr("__dict__");
    bp::exec_file(argv[1], interpreter, interpreter);

    int port = bp::extract<int>(interpreter["port"]);
    std::string desc =
        bp::extract<std::string>(interpreter["desc"]);
    std::cout << "Starting server " << desc
        << " on port " << port;
}
```

From the c++ side

```
namespace bp = boost::python;

int main(int argc, char** argv)
{
    Py_Initialize();
    PyEval_InitThreads();
    bp::object main = bp::import("__main__");
    bp::object interpreter = main.attr("__dict__");
    bp::exec_file(argv[1], interpreter, interpreter);

    int port = bp::extract<int>(interpreter["port"]);
    std::string desc =
        bp::extract<std::string>(interpreter["desc"]);
    std::cout << "Starting server " << desc
        << " on port " << port;
}
```

From the c++ side

```
namespace bp = boost::python;

int main(int argc, char** argv)
{
    Py_Initialize();
    PyEval_InitThreads();
    bp::object main = bp::import("__main__");
    bp::object interpreter = main.attr("__dict__");
    bp::exec_file(argv[1], interpreter, interpreter);

    int port = bp::extract<int>(interpreter["port"]);
    std::string desc =
        bp::extract<std::string>(interpreter["desc"]);
    std::cout << "Starting server " << desc
        << " on port " << port;
}
```

Get the next relevant file

```
import glob, os, sys

# glob together files I like at startup
files_i_like = []
for f in glob.glob("/path/to/data/*.i3.gz"):
    if meets_some_criteria(f):
        files_i_like.append(f)

description = "Some server"
port = 26227

i = 0
def next():
    global i
    if i > len(files_i_like):
        return None
    i += 1
    return files_i_like[i-1]
```

Get the next relevant file

```
import glob, os, sys

# glob together files I like at startup
files_i_like = []
for f in glob.glob("/path/to/data/*.i3.gz"):
    if meets_some_criteria(f):
        files_i_like.append(f)

description = "Some server"
port = 26227

i = 0
def next():
    global i
    if i > len(files_i_like):
        return None
    i += 1
    return files_i_like[i-1]
```

Get the next relevant file

```
import glob, os, sys

# glob together files I like at startup
files_i_like = []
for f in glob.glob("/path/to/data/*.i3.gz"):
    if meets_some_criteria(f):
        files_i_like.append(f)

description = "Some server"
port = 26227

i = 0
def next():
    global i
    if i > len(files_i_like):
        return None
    i += 1
    return files_i_like[i-1]
```

Get the next relevant file

```
import glob, os, sys

# glob together files I like at startup
files_i_like = []
for f in glob.glob("/path/to/data/*.i3.gz"):
    if meets_some_criteria(f):
        files_i_like.append(f)

description = "Some server"
port = 26227

i = 0
def next():
    global i
    if i > len(files_i_like):
        return None
    i += 1
    return files_i_like[i-1]
```

Get the next relevant file

```
import glob, os, sys

# glob together files I like at startup
files_i_like = []
for f in glob.glob("/path/to/data/*.i3.gz"):
    if meets_some_criteria(f):
        files_i_like.append(f)

description = "Some server"
port = 26227

i = 0
def next():
    global i
    if i > len(files_i_like):
        return None
    i += 1
    return files_i_like[i-1]
```

Get the next relevant file

```
import glob, os, sys

# glob together files I like at startup
files_i_like = []
for f in glob.glob("/path/to/data/*.i3.gz"):
    if meets_some_criteria(f):
        files_i_like.append(f)

description = "Some server"
port = 26227

i = 0
def next():
    global i
    if i > len(files_i_like):
        return None
    i += 1
    return files_i_like[i-1]
```

Get the next relevant file

```
import glob, os, sys

# glob together files I like at startup
files_i_like = []
for f in glob.glob("/path/to/data/*.i3.gz"):
    if meets_some_criteria(f):
        files_i_like.append(f)

description = "Some server"
port = 26227

i = 0
def next():
    global i
    if i > len(files_i_like):
        return None
    i += 1
    return files_i_like[i-1]
```

From the c++ side

```
Py_Initialize();
PyEval_InitThreads();
bp::object main = bp::import("__main__");
bp::object interpreter = main.attr("__dict__");
bp::exec_file(argv[1], interpreter, interpreter);

int port = bp::extract<int>(interpreter["port"]);

std::string desc =
    bp::extract<std::string>(interpreter["description"]);

bp::object next = interpreter["next"];
for (bp::object o = next(); o.ptr() != Py_None; o = next())
{
    std::string filename = bp::extract<std::string>(o);
    handle(filename);
}
```

From the c++ side

```
Py_Initialize();
PyEval_InitThreads();
bp::object main = bp::import("__main__");
bp::object interpreter = main.attr("__dict__");
bp::exec_file(argv[1], interpreter, interpreter);

int port = bp::extract<int>(interpreter["port"]);

std::string desc =
    bp::extract<std::string>(interpreter["description"]);

bp::object next = interpreter["next"];
for (bp::object o = next(); o.ptr() != Py_None; o = next())
{
    std::string filename = bp::extract<std::string>(o);
    handle(filename);
}
```

From the c++ side

```
Py_Initialize();
PyEval_InitThreads();
bp::object main = bp::import("__main__");
bp::object interpreter = main.attr("__dict__");
bp::exec_file(argv[1], interpreter, interpreter);

int port = bp::extract<int>(interpreter["port"]);

std::string desc =
    bp::extract<std::string>(interpreter["description"]);

bp::object next = interpreter["next"];
for (bp::object o = next(); o.ptr() != Py_None; o = next())
{
    std::string filename = bp::extract<std::string>(o);
    handle(filename);
}
```

From the c++ side

```
Py_Initialize();
PyEval_InitThreads();
bp::object main = bp::import("__main__");
bp::object interpreter = main.attr("__dict__");
bp::exec_file(argv[1], interpreter, interpreter);

int port = bp::extract<int>(interpreter["port"]);

std::string desc =
    bp::extract<std::string>(interpreter["description"]);

bp::object next = interpreter["next"];
for (bp::object o = next(); o.ptr() != Py_None; o = next())
{
    std::string filename = bp::extract<std::string>(o);
    handle(filename);
}
```

From the c++ side

```
Py_Initialize();
PyEval_InitThreads();
bp::object main = bp::import("__main__");
bp::object interpreter = main.attr("__dict__");
bp::exec_file(argv[1], interpreter, interpreter);

int port = bp::extract<int>(interpreter["port"]);

std::string desc =
    bp::extract<std::string>(interpreter["description"]);

bp::object next = interpreter["next"];
for (bp::object o = next(); o.ptr() != Py_None; o = next())
{
    std::string filename = bp::extract<std::string>(o);
    handle(filename);
}
```