

# Intro to IceTray

## Madison Bootcamp 2020

Alex Olivas

# IceTray : The Definitive Guide

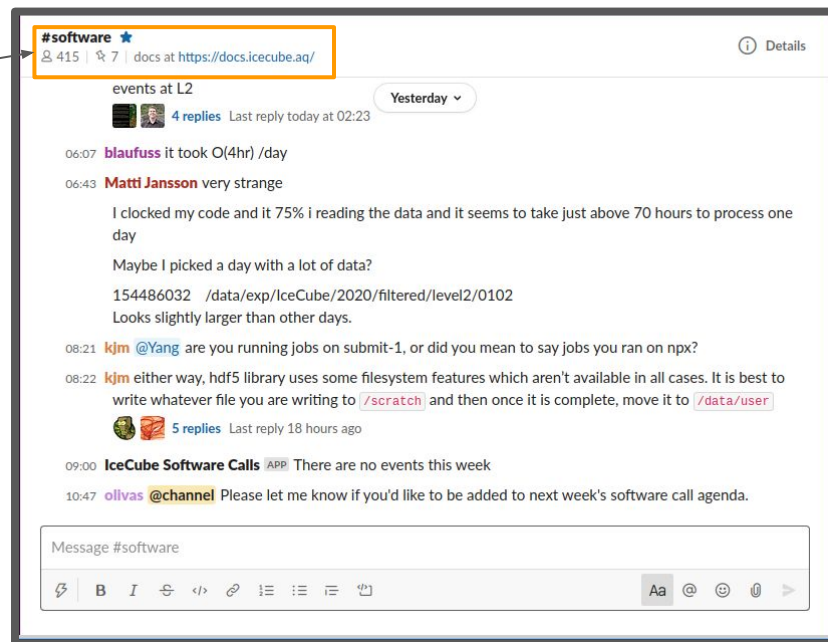
The topic of the **#software** channel will always point to the documentation builds.

<https://docs.icecube.aq>

(100+ projects through L2/L3)

Now contains builds of the trunk as well as past releases.

(Thanks D. Schultz!)



If you see something, say something.

<http://code.icecube.wisc.edu/projects/icecube/newticket>

# The Ticketing System

<http://code.icecube.wisc.edu>

IceCube Software - Chromium

code.icecube.wisc.edu/projects/icecube

ICECUBE

Wiki Timeline Roadmap Browse Source View Tickets **New Ticket** Logout Preferences Help/Guide About Trac

Search Admin Ticket Graph

wiki: WikiStart

### Welcome to the IceCube Trac system

Welcome to the IceCube Software Repository. From this page, you may:

- [Browse source code with Trac](#)
- [Browse source code with simple SVN](#)

**Please make sure your name and email address are properly set** for your account on the [Preferences](#). This will allow us to correspond with you regarding feature requests and bug reports, and enables you to retrieve your password email should you need to. And please, use an email account from an IceCube institution.

If you have any problems, requests for new features, or other comments, please send them to [svn\\_admin \(at\) icecube.umd.edu](mailto:svn_admin@icecube.umd.edu)

### Documentation

Documentation has its own dedicated site at <http://software.icecube.wisc.edu/>.

### Discussion

Please join us for software related discussions on [Slack](#). You can find us find us in the [IceCube-SPNO team](#) in the channel, [#software](#). If you've never used Slack before, please see the [IceCube wiki page about Slack](#).

### Testing

We build and test on various platforms. [Build results](#) are available. If you desire a new automated build/test platform, please, email us at [svn\\_admin \(at\) icecube.umd.edu](mailto:svn_admin@icecube.umd.edu). List of bots is <http://builds.icecube.wisc.edu/buildslaves>.

### General Trac Help

Trac is a **minimalistic** approach to **web-based** management of **software projects**. Its goal is to simplify effective tracking and handling of software issues, enhancements and overall progress.

- [TracGuide](#) -- Built-in Documentation
- [Trac FAQ](#) -- Frequently Asked Questions
- [TracSupport](#) -- Trac Support

For a complete list of local wiki pages, see [TitleIndex](#).

Last modified 2 years ago

Edit this page Attach file Rename page Delete this version Delete page

Download in other formats: Plain Text

Powered by Trac 1.0.1 By Edgewall Software

You are welcome to visit our Web Site <http://icecube.umd.edu>

Creating a new ticket is very easy.

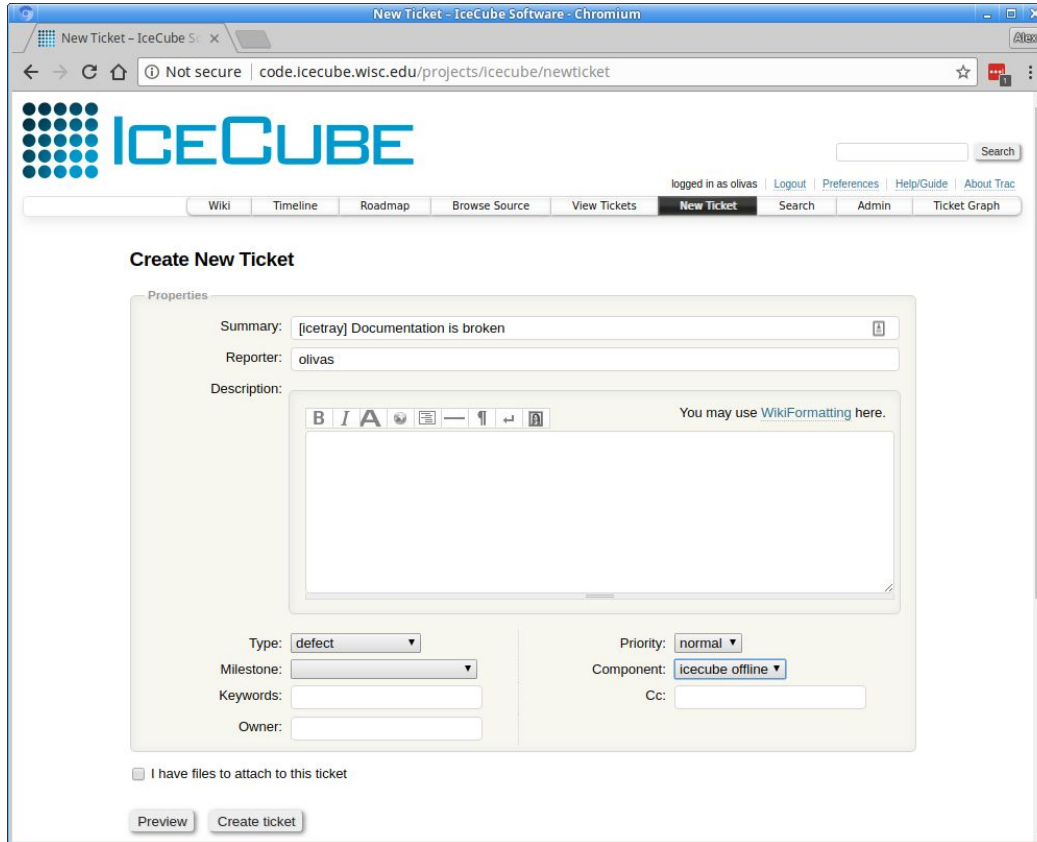
All ticket changes are reported in #software on Slack.

Please, please, please file a ticket if you find issues (including documentation).

We can't fix problems we don't know about.

# The Ticketing System

<http://code.icecube.wisc.edu>



The screenshot shows a web browser window titled "New Ticket - IceCube Software - Chromium" with the URL "code.icecube.wisc.edu/projects/icecube/newticket". The page features the IceCube logo and a navigation menu with options like "Wiki", "Timeline", "Roadmap", "Browse Source", "View Tickets", "New Ticket", "Search", "Admin", and "Ticket Graph". The "Create New Ticket" form is the central focus, containing the following fields and options:

- Summary:** [icetray] Documentation is broken
- Reporter:** olivas
- Description:** A text area with a rich text editor toolbar (bold, italic, align, link, unlink, list, ul, indent, outdent, undo, redo) and a note: "You may use WikiFormatting here."
- Type:** defect
- Priority:** normal
- Milestone:** (empty dropdown)
- Component:** icecube offline
- Keywords:** (empty text field)
- Owner:** (empty text field)
- Cc:** (empty text field)

At the bottom of the form, there is a checkbox labeled "I have files to attach to this ticket" and two buttons: "Preview" and "Create ticket".

One and only one hard and fast rule:

**Do not submit tickets as 'icecube'**

Fill in the fields as best you can.

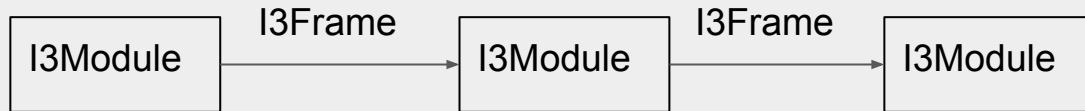
Convention: In the description, it's helpful, but not necessary, to start with the project name in square brackets.

# IceTray : A Very Brief Introduction

## I3Tray

**IceTray is the framework whose responsibility is to manage interactions between user-defined I3Modules, passing I3Frames from module to module.**

- I3Frame - Data Container
- I3Module - Take data out of the frame, process, add data to the frame.



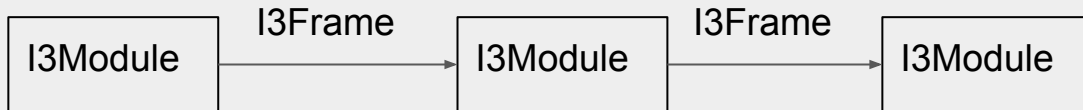
# IceTray : The I3Frame and I3Module

## I3Tray

**I3Frame - Dictionary with string keys and "frame objects" values.**

\*Trivia : Unlike true python dictionaries, which can store objects of any type, I3Frames can only contain objects which inherit from I3FrameObject. Driven by C++ design.

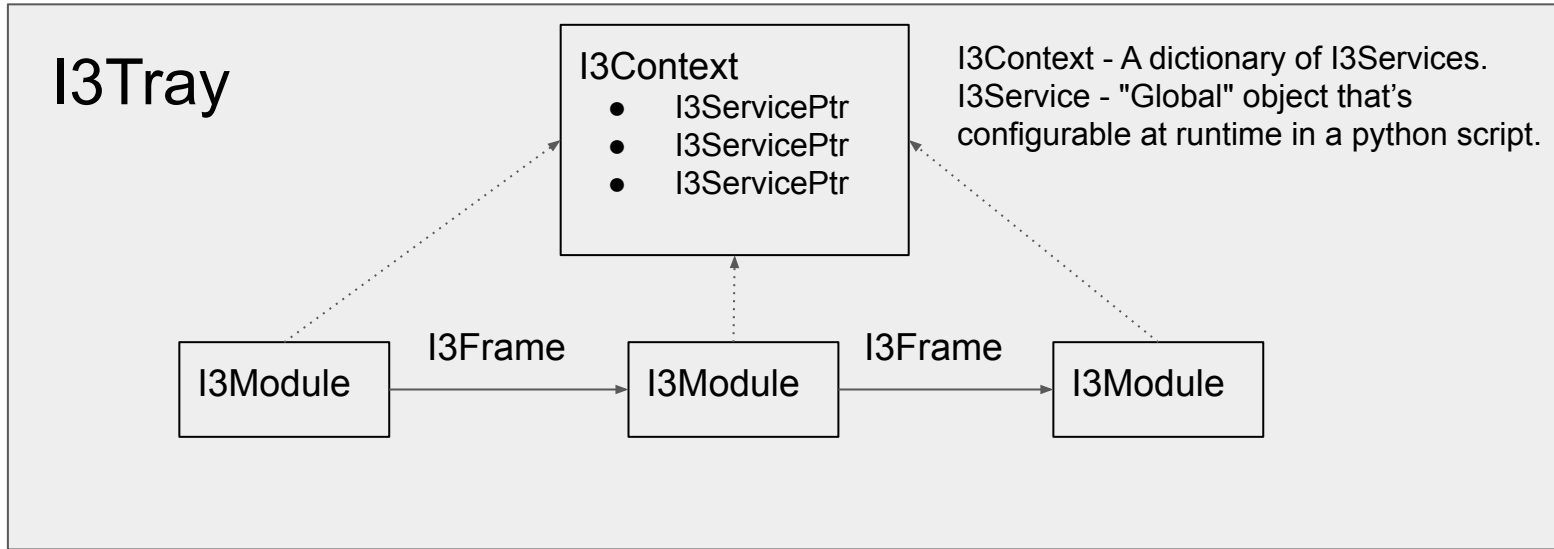
\*Trivia : I3Frame is actually a C++ class which manages `map<string, shared_ptr<I3FrameObject>>`



Nearly all I3FrameObjects are collected in three projects: dataclasses, simclasses, recclasses.

- dataclasses - <https://docs.icecube.aq/combo/trunk/projects/dataclasses/index.html>
- simclasses - <https://docs.icecube.aq/combo/trunk/projects/simclasses/index.html>
- recclasses - <https://docs.icecube.aq/combo/trunk/projects/recclasses/index.html>

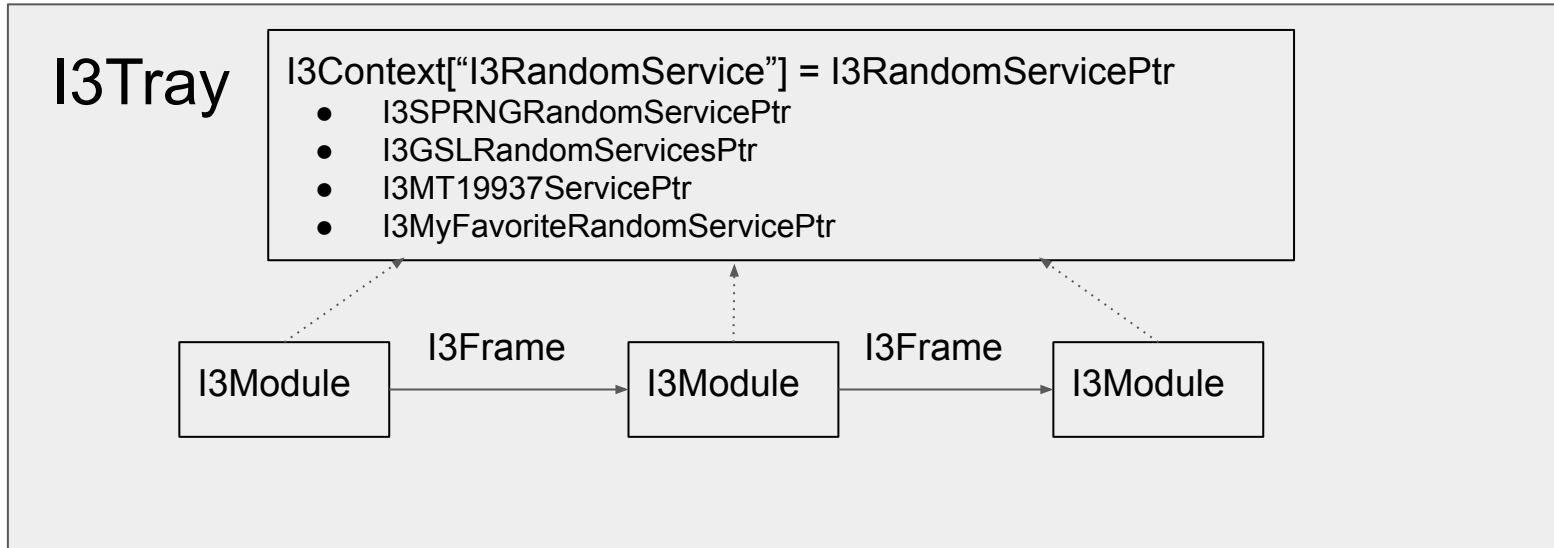
# IceTray : I3Context and I3Services



When to use a proper I3Service stored in the I3Context?

- "Global" object used by several (**lots**) I3Modules

# IceTray : RNG Service Example



User can choose any random service they want at runtime and no downstream module needs to change.



# IceTray : The Frame-Stream-Stop Model

Frames come in different flavors

## I3Frame Types

I3Frame::TrayInfo

I3Frame::Geometry

I3Frame::Calibration

I3Frame::DetectorStatus

I3Frame::Physics

I3Frame::DAQ

## I3Module 'Stops'

Stops are methods that correspond to a frame type.

I3Module::Geometry

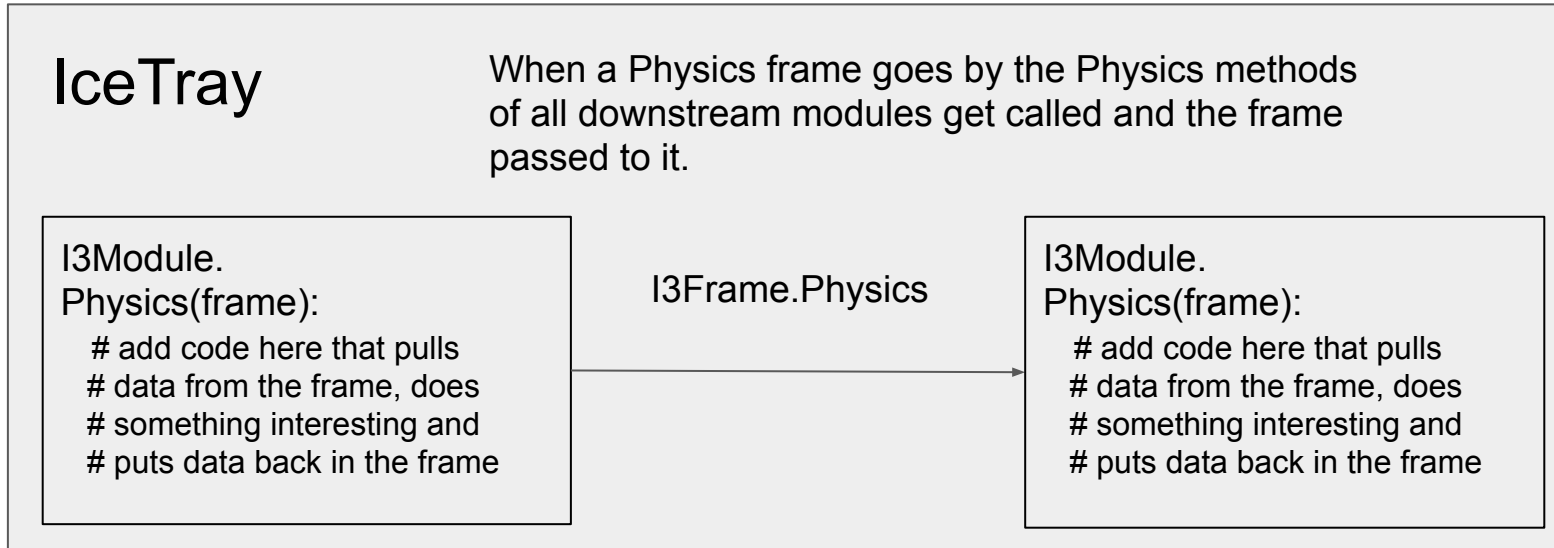
I3Module::Calibration

I3Module::DetectorStatus

I3Module::Physics

I3Module::DAQ

# IceTray : The Frame-Stream-Stop Model



# PyTray : IceTray in Working Pseudo-Code

```
class I3Service(object):
    pass

class I3FrameObject(object):
    pass

class I3Frame(object):
    def __init__(self, frame_type='P'):
        self.frame_type = frame_type
        self.state = dict()

    def __getitem__(self, key):
        return self.state[key]

    def __setitem__(self, key, value):
        if isinstance(value, I3FrameObject):
            self.state[key] = value
        else:
            raise TypeError("%s is not an I3FrameObject" % key)

    def __str__(self):
        result = '[ I3Frame (%s) \n' % self.frame_type
        result += "".join([' %s: %s' % (k,v) for k,v in self.state.items()])
        result += '\n]'
        return result
```

I3Service and I3FrameObject serve simply as a base class to mimic C++ design and behavior.

I3Frame essentially wraps a dict.

Only objects that inherit from I3FrameObject can be added to the I3Frame.

# PyTray : IceTray in Working Pseudo-Code

```
class I3Tray(object):
    def __init__(self):
        self.context = dict()
        self.__modules = list()

    def Add(self, obj, name, **kwargs):
        """
        Adds I3Modules and I3Services to the framework.
        """
        self.__add(obj, name, **kwargs)

    def Execute(self):
        """
        Configures the modules and then executes them.
        in the order they were added.
        """
        self.__execute()
```

# PyTray : IceTray in Working Pseudo-Code


```
def __add(self, obj, name, **kwargs):  
    if isinstance(obj, I3Service):  
        self.context[name] = obj(**kwargs)  
    elif isinstance(obj, I3Module):  
        self.__modules.append(obj(self.context, **kwargs))  
    elif callable(obj):  
        # support only function objects for now  
        print(kwargs)  
        self.__modules.append(obj(self.context, **kwargs))  
    else:  
        raise TypeError("": %s" % name)
```

Classes that inherit from I3Service go in the tray's context.

# PyTray : IceTray in Working Pseudo-Code

```
def __add(self, obj, name, **kwargs):  
    if isinstance(obj, I3Service):  
        self.context[name] = obj(**kwargs)  
    elif isinstance(obj, I3Module):  
        self.__modules.append(obj(self.context, **kwargs))  
    elif callable(obj):  
        # support only function objects for now  
        print(kwargs)  
        self.__modules.append(obj(self.context, **kwargs))  
    else:  
        raise TypeError("": %s" % name)
```

I3Modules and function objects are created with a context and appended to an internal private list.



# PyTray : IceTray in Working Pseudo-Code

```
def __execute(self):
```

```
    for module in self.__modules:
        if hasattr(module, 'Configure'):
            module.Configure()
```

First call the configure method of each module.  
The order shouldn't matter.

```
    while True:
```

```
        frame = self.__modules[0].GenerateFrame()
        if not frame:
            break
```

In an infinite loop, grab the first frame from the  
"Driving Module" The driving module returns  
**None** when it's done.

```
        for module in self.__modules:
            if callable(module):
                if not module(frame):
                    continue
            elif frame.frame_type == 'G':
                if not module.Geometry(frame):
                    continue
            elif frame.frame_type == 'C':
                if not module.Calibration(frame):
                    continue
            elif frame.frame_type == 'D':
                if not module.DetectorStatus(frame):
                    continue
            elif frame.frame_type == 'Q':
                if not module.DAQ(frame):
                    continue
            elif frame.frame_type == 'P':
                if not module.Physics(frame):
                    continue
            else:
                if not module.Default(frame):
```

Depending on the "frame type" call the  
corresponding stop, in the order they were  
added to the tray.

If any 'Stop' returns False go to the next frame

If it's a user-defined frame (e.g. 'M','S','W',  
etc...) just call the Default method.

# PyTray : IceTray in Working Pseudo-Code

```
class I3Module(object):
    """
    Base class for IceTray modules.
    """
    def __init__(self, context):
        self.context = context

    def Configure(self):
        pass

    def GenerateFrame(self):
        """
        This is only called if it's first in the list.
        It's the job of the 'Driving Module' to create frames.
        """
        raise NotImplementedError

    def Geometry(self, frame):
        return True

    def Calibration(self, frame):
        return True

    def DetectorStatus(self, frame):
        return True

    def DAQ(self, frame):
        return True

    def Physics(self, frame):
        return True

    def Default(self, frame):
        return True
```

Classes that inherit from I3Module can choose to implement any 'Stop' they want.

On construction (by I3Tray) modules will receive and have access to the context.

Configure is called just before execution in I3Tray.Execute.

Return True if you want the next module to be able to process the frame as well.



# PyTray : IceTray in Working Pseudo-Code

```
class I3Source(I3Module):
    def __init__(self, context, n_frames, frame_type):
        super(I3Source, self).__init__(context)
        self.n_frames = n_frames
        self.frame_type = frame_type
        self.n_frames_served = 0

    def GenerateFrame(self):
        if self.n_frames_served < self.n_frames:
            self.n_frames_served += 1
            return I3Frame(self.frame_type)

class Dump:
    def __init__(self, context):
        self.context = context
        self.frame_counter = 0

    def __call__(self, frame):
        self.frame_counter += 1
        print("Frame Counter = %d" % self.frame_counter)
        print(frame)
```

```
class I3Writer:
    def __init__(self, context, filename):
        self.context = context
        self.filename = filename

    def __call__(self, frame):
        with open(self.filename, 'w') as f:
            pickle.dump({'type' : frame.frame_type,
                        'state' : frame.state},
                        f)

if __name__ == '__main__':

    tray = I3Tray()
    tray.Add(I3Source, 'source', n_frames=100, frame_type='Q')
    tray.Add(Dump, 'dump')
    tray.Add(I3Writer, 'writer', filename='output.pkl')
    tray.Execute()
```

# IceTray : I3FrameObjects

```
Terminal - IPython: fresh_trunk/build
File Edit View Terminal Tabs Help
[ 9:00AM ] [ olivas@finn:~/icecube/combo/fresh_trunk/build ]
$ ipython
Python 2.7.12 (default, Dec 4 2017, 14:50:18)
Type "copyright", "credits" or "license" for more information.

IPython 5.5.0 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.

In [1]: from icecube import icetray, dataclasses

In [2]: frame = icetray.I3Frame()

In [3]: frame['my_tree'] = dataclasses.I3MCTree()

In [4]: frame['not_a_frame_object'] = [1,2,3]

ArgumentError                                Traceback (most recent call last)
<ipython-input-4-158b928ffc76> in <module>()
----> 1 frame['not_a_frame_object'] = [1,2,3]

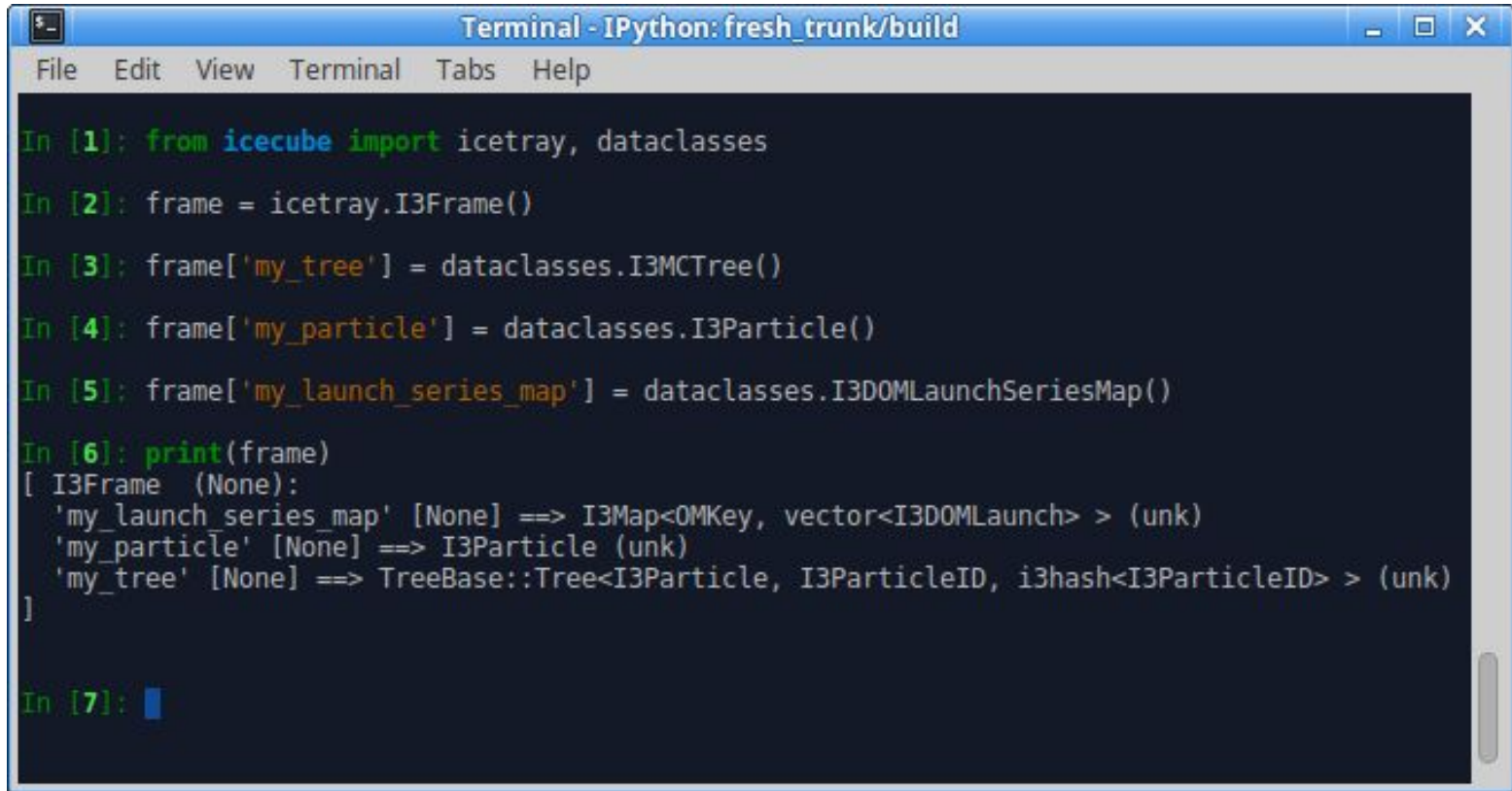
ArgumentError: Python argument types in
  I3Frame._setitem_(I3Frame, str, list)
did not match C++ signature:
  _setitem_(I3Frame {lvalue}, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, boost::shared_ptr<I3FrameObject const>)

In [5]:
```

I3MCTree "is-a" (i.e. inherits from) I3FrameObject, found in dataclasses.

A list of numbers is not an I3FrameObject.

# IceTray : I3FrameObjects

A terminal window titled "Terminal - IPython: fresh\_trunk/build" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows a series of IPython commands and their output. The commands create an I3Frame object and populate it with three dataclasses: I3MCTree, I3Particle, and I3DOMLaunchSeriesMap. The final command prints the frame object, showing its structure and the objects it contains.

```
Terminal - IPython: fresh_trunk/build
File Edit View Terminal Tabs Help

In [1]: from icecube import icetray, dataclasses

In [2]: frame = icetray.I3Frame()

In [3]: frame['my_tree'] = dataclasses.I3MCTree()

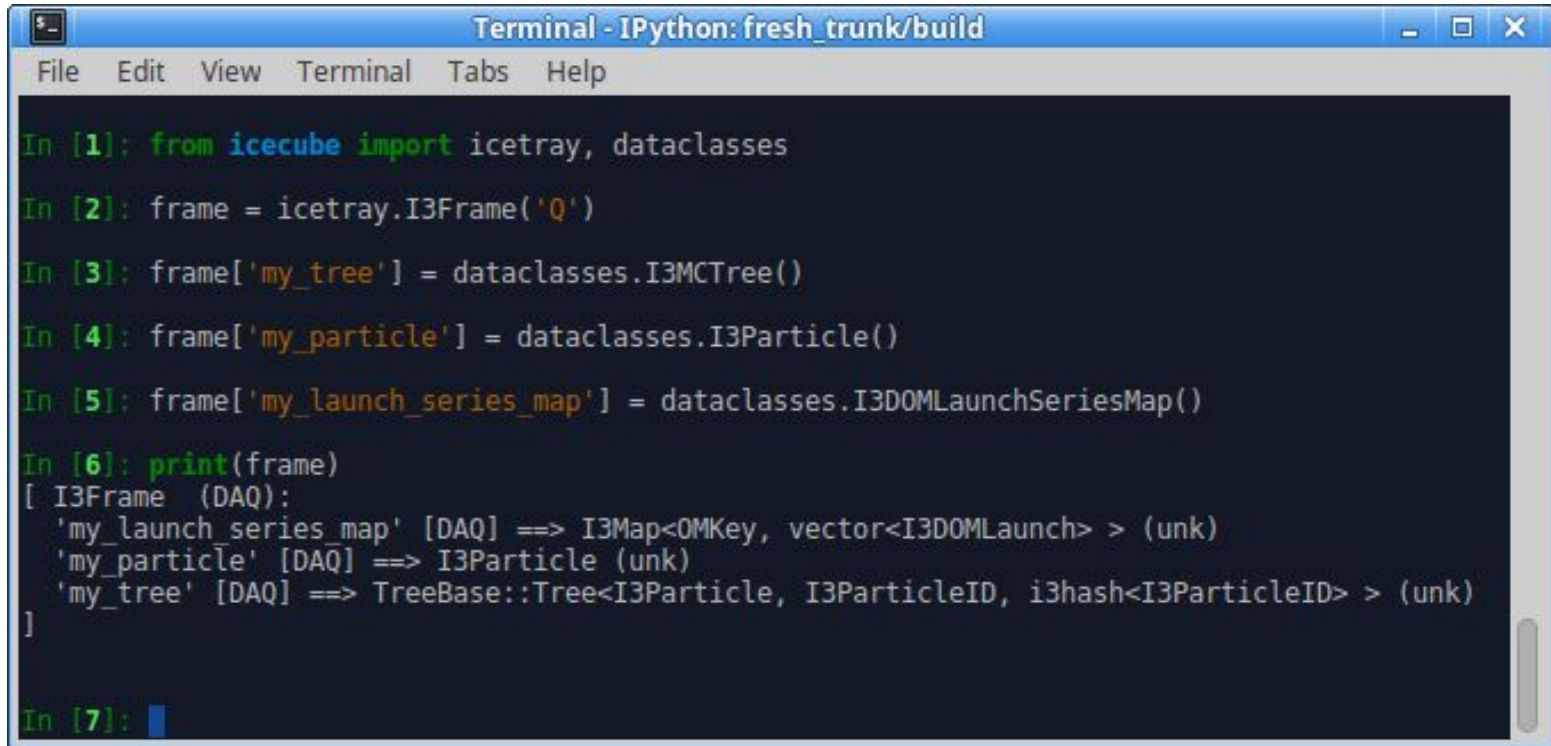
In [4]: frame['my_particle'] = dataclasses.I3Particle()

In [5]: frame['my_launch_series_map'] = dataclasses.I3DOMLaunchSeriesMap()

In [6]: print(frame)
[ I3Frame (None):
  'my_launch_series_map' [None] ==> I3Map<OMKey, vector<I3DOMLaunch> > (unk)
  'my_particle' [None] ==> I3Particle (unk)
  'my_tree' [None] ==> TreeBase::Tree<I3Particle, I3ParticleID, i3hash<I3ParticleID> > (unk)
]

In [7]: █
```

# IceTray : Creating Typed Frames

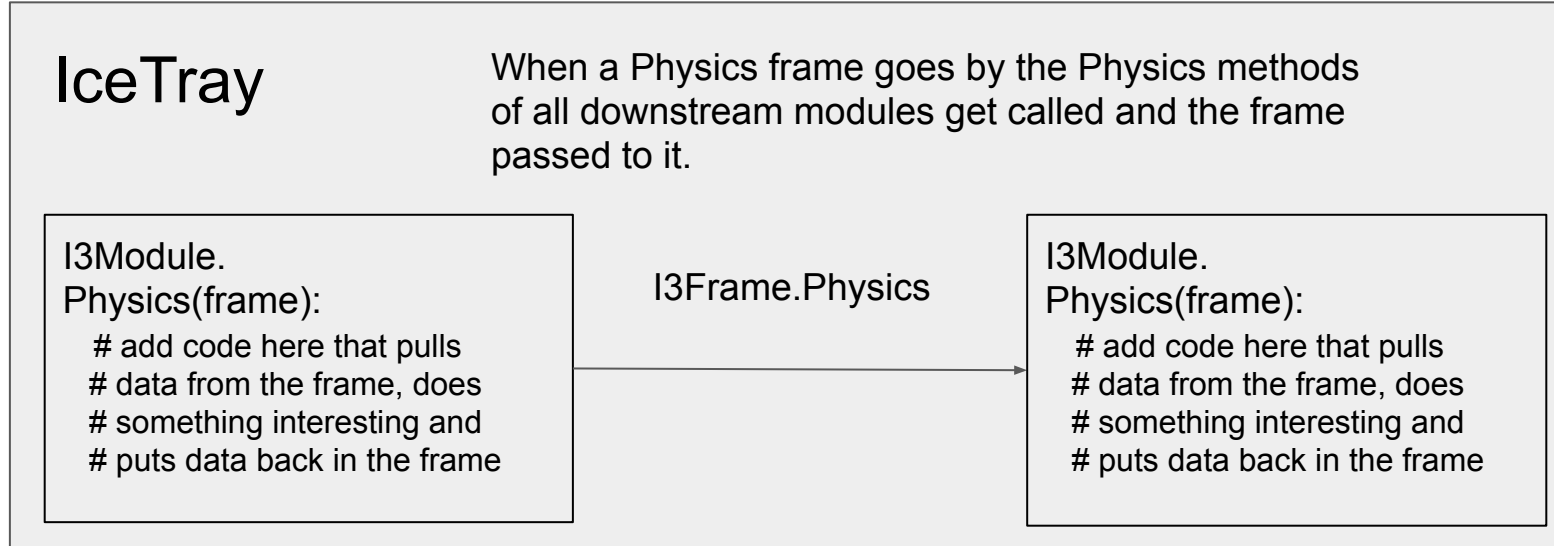


```
Terminal - IPython: fresh_trunk/build
File Edit View Terminal Tabs Help

In [1]: from icecube import icetray, dataclasses
In [2]: frame = icetray.I3Frame('Q')
In [3]: frame['my_tree'] = dataclasses.I3MCTree()
In [4]: frame['my_particle'] = dataclasses.I3Particle()
In [5]: frame['my_launch_series_map'] = dataclasses.I3DOMLaunchSeriesMap()
In [6]: print(frame)
[ I3Frame (DAQ):
  'my_launch_series_map' [DAQ] ==> I3Map<OMKey, vector<I3DOMLaunch> > (unk)
  'my_particle' [DAQ] ==> I3Particle (unk)
  'my_tree' [DAQ] ==> TreeBase::Tree<I3Particle, I3ParticleID, i3hash<I3ParticleID> > (unk)
]

In [7]:
```

# IceTray : The Frame-Stream-Stop Model



Frame Hierarchy - Generally expected in this order: **GCDQP**

Frame Mixing - Example: All objects from G, C, D, Q are accessible from P-frames.

Frame Packets - **QPPPPP** P-frames following Q-frame "belong" to the Q-frame.

`I3PacketModule` - Allows you to process a vector of frames (i.e. "packet").

# IceTray : Frame Splitting

## Original IceTray Stream (pre-Q-frames)

G  
C  
D  
PPPPPPPPPPP...

## Introduction of Q (DAQ frames)

Driven by SLOP Trigger (Slow Monopole)

Slow Monopoles can take ms to traverse the detector

G  
C  
D  
Q Q Q Q ...  
P P P P ...

Q Frames would contain Triggered (i.e. DAQ) information.  
The reconstructions would stay in P frames.

What about the many muons now embedded in a single Q/P frame? Lots of reconstruction modules had already been written assuming a single particle and work great w/ SMT8.

# IceTray : Frame Splitting

Original IceTray Stream (pre-Q-frames)

G  
C  
D  
PPPPPPPPPPP...

Introduction of Q (DAQ frames)

Driven by SLOP (if memory serves)

Slow Monopoles can take ms to traverse the detector

G  
C  
D  
Q        Q        Q        Q

<https://docs.icecube.aq/combo/trunk/projects/trigger-splitter/index.html>

I3TriggerSplitter splits single P-frames depending on the desired trigger.

PPPPP    PPPPP    PPPPP    PPPPPPPPPPPPPPP

# IceTray : Frame Packets and I3PacketModule

## Introduction of Q (DAQ frames)

Driven by SLOP

Slow Monopoles can take ms to traverse the detector

G  
C  
D  
Q Q  
PPPPP PPPPPP

Potentially essential for anyone wanting to deal with InIce/IceTop reconstructions.

```
class I3PacketModule : public I3Module
{
public:
    I3PacketModule(const I3Context& context,
                  I3Frame::Stream sentinel = I3Frame::DAQ);
    ~I3PacketModule();

    void Configure_();
    void Process();
    void FlushQueue();
    void Finish();

    virtual void FramePacket(std::vector<I3FramePtr> &packet);

protected:
    I3Frame::Stream sentinel;
    std::vector<I3Frame::Stream> packet_types;

private:
    std::vector<I3FramePtr> queue_;
    boost::python::object if_;
};
```



# IceTray : Tray Segments

Segments can contain multiple I3Modules to help bundle code together.

Let's use a segment:

```
from icecube import payload_parsing
tray.Add(payload_parsing.I3DOMLaunchExtractor)
```

Writing a segment:

```
from icecube import icetray
@icetray.traysegment
def MySegment(tray, name, arg1, If = lambda f:True, **kwargs):
    # we can use arg1 or the dict of kwargs
    tray.Add("Dump", If=If)
```

# TriggerSim Segment

## TriggerSim Segment

The TriggerSim segment includes the following modules :

- SimpleMajorityTrigger
- ClusterTrigger
- CylinderTrigger
- SlowMonopoleTrigger
- I3GlobalTriggerSim
- I3Pruner
- I3TimeShifter

All of the above are added conditionally, with the exception of the I3GlobalTriggerSim, which is always included. The trigger modules : SimpleMajorityTrigger, ClusterTrigger, CylinderTrigger, and SlowMonopoleTrigger will only be added if there's a configuration in the GCD file. The I3Pruner and I3TimeShifter can be disabled via segment parameters, but it's not advised. You should really know what you're doing and the purpose they serve before disabling them.

## Parameters

- **tray** - Standard for segments.
- **name** - Standard for segments.
- **gcd\_file** - The GCD (I3File) that contains the trigger configuration. Note the segment figures out from the GCD file which trigger modules need to be loaded and configures them accordingly.
- **prune** (DEFAULT = True) - Whether to remove launches outside the readout windows. Nearly everyone will want to keep this set at True. It makes simulation look more like data.
- **time\_shift** (DEFAULT = True) - Whether to time shift time-like frame objects. Nearly everyone will want to keep this set at True. It makes simulation look more like data.
- **time\_shift\_args** (DEFAULT = dict()) - dict that's forwarded to the I3TimeShifter module. See below for more details.
- **filter\_mode** (DEFAULT = True) - Whether to filter frames that do not trigger.

IceCube triggering system applies four independent triggering algorithms.

It combines the results to generate a global trigger, removes launches outside the readout window, and shifts the time of the objects to make them look like data.

You can re-trigger your data the "standard" way with two lines in your script:

```
from icecube.trigger_sim import TriggerSim
...
tray.Add(TriggerSim, gcd_file = dataio.I3File(<path_to_GCD>))
```

# IceTray History Lesson

**From 2008 - Present...**

Why? Because lots of production scripts haven't been updated in more than 12 years.

\*Still lots of production scripts and segments that haven't been cleaned up in over 12 years.

# IceTray : Pre-2008

```
import sys
from I3Tray import *
load('libicetray')
load('libdataclasses')
load('libdataio')

tray = I3Tray()
tray.AddModule('I3Reader', 'reader') (('Filenamelist', sys.argv[1:]))
tray.AddModule('Dump', 'dumper')
tray.AddModule('TrashCan', 'can')
tray.Execute()
tray.Finish()
```

Very non-pythonic.

- Call 'load' explicitly.
- Odd AddModule signature.

# IceTray : Pre-2012

```
import sys
from I3Tray import *
from icecube import icetray, dataclasses, dataio, phys_service
```

Import IceCube projects the python way.

```
tray = I3Tray()
tray.AddService('I3GSLRandomServiceFactory', 'gsl', Seed=42)
tray.AddModule('I3Reader', 'reader', Filenamelist=sys.argv[1:])
tray.AddModule('Dump', 'dumper')
tray.AddModule('TrashCan', 'can')
tray.Execute()
tray.Finish()
```

Pythonic signature with keyword arguments.

# IceTray : Post-2013

## Cleanups

- Just 'Add'
- Anonymous I3Modules - No need to include a name.
- No need to add “TrashCan” Module.
- Need to call “Finish” explicitly.

```
import sys
from I3Tray import *
from icecube import icetray, dataclasses, dataio, phys_service

tray = I3Tray()
tray.Add('I3GSLRandomServiceFactory', 'gsl', Seed=42)
tray.Add('I3Reader', Filenamelist=sys.argv[1:])
tray.Add('Dump')
tray.Execute()
```

# The Simplest IceTray Chain

I3InfiniteSource is a C++ module located in the dataio project.

```
#!/usr/bin/env python
from I3Tray import I3Tray
from icecube import dataio

tray = I3Tray()
tray.Add("I3InfiniteSource")
tray.Add("Dump")
tray.Execute(10)
```

## IceTray : Functions as IceTray Modules

```
#!/usr/bin/env python
from I3Tray import I3Tray
from icecube import icetray, dataio, dataclasses

def generator(frame):
    frame["tree"] = dataclasses.I3MCTree()

tray = I3Tray()
tray.Add("I3InfiniteSource")
tray.Add(generator, streams = [icetray.I3Frame.DAQ])
tray.Add("Dump")
tray.Execute(10)
```



# IceTray : Lambda as 'Modules'

Very simple filter.

```
#!/usr/bin/env python
from I3Tray import I3Tray
from icecube import icetray, dataio, dataclasses

def generator(frame):
    frame["tree"] = dataclasses.I3MCTree()

tray = I3Tray()
tray.Add("I3InfiniteSource")
tray.Add(generator, streams = [icetray.I3Frame.DAQ])
tray.Add(lambda frame : frame.Has("tree"), streams = [icetray.I3Frame.DAQ])
tray.Add("Dump")
tray.Execute(10)
```

If function returns True, the frame is passed to the next module.

# Tray Segments

Group several 'modules' and functions together to form something that can plug into IceTray.

```
#!/usr/bin/env python
from I3Tray import I3Tray
from iccube import icetray, dataio, dataclasses

@icetray.traysegment
def GeneratorSegment(tray, name):
    def generator(frame):
        frame["tree"] = dataclasses.I3MCTree()

    tray.Add("I3InfiniteSource")
    tray.Add(generator, streams = [icetray.I3Frame.DAQ])
    tray.Add(lambda frame : frame.Has("tree"),
             streams = [icetray.I3Frame.DAQ])

tray = I3Tray()
tray.Add(GeneratorSegment)
tray.Add("Dump")
tray.Execute(10)
```

## I3Module : Post-Modern Classic - Pre-2017

```
class ExampleModule(icetray.I3Module):
    def __init__(self, context):
        icetray.I3Module.__init__(self, context)
        self.AddOutBox("OutBox")

    def Configure(self):
        pass
```

# I3Module : Post-Modern Classic - Post-2017

```
class ExampleModule(icetray.I3Module):  
    def __init__(self, context):  
        icetray.I3Module.__init__(self, context)
```

**This is now the simplest IceTray module that works, but does absolutely nothing useful.**

You don't have to implement a Configure method if it doesn't need one.

You don't have to explicitly add an OutBox anymore. The default works just fine for >99% of modules in production.

# I3Module :

Post-Modern Classic

Example of a fully-working icetray chain that does absolutely nothing.

The best we can say about this is that it will execute without throwing.

Simplest illustration of most concepts up to this point.

```
#!/usr/bin/env python
from I3Tray import I3Tray
from icecube import icetray, dataio, dataclasses

@icetray.traysegment
def GeneratorSegment(tray, name):
    def generator(frame):
        frame["tree"] = dataclasses.I3MCTree()

    tray.Add("I3InfiniteSource")
    tray.Add(generator, streams = [icetray.I3Frame.DAQ])
    tray.Add(lambda frame : frame.Has("tree"),
             streams = [icetray.I3Frame.DAQ])

class ExampleModule(icetray.I3Module):
    def __init__(self, context):
        icetray.I3Module.__init__(self, context)

tray = I3Tray()
tray.Add(GeneratorSegment)
tray.Add(ExampleModule)
tray.Add("Dump")
tray.Execute(10)
```

# I3Module: Parameters

Parameters defined with 'AddParameter' become keyword arguments when added to an I3Tray instance.

```
from icecube import icetray

class ExampleModule(icetray.I3Module):
    def __init__(self, context):
        icetray.I3Module.__init__(self, context)
        self.default_param_value = 42
        self.AddParameter("SomeParam", "Docstring...", default_param_value)

    def Configure(self):
        self.some_param = self.GetParameter("SomeParam")
        if self.some_param != self.default_param_value:
            print("User changed SomeParam to %d" % self.some_param)

    def DAQ(self, frame):
        print("Running with SomeParam = %d" % self.some_param)

tray = I3Tray()
tray.Add(ExampleModule, some_param = 32)
```

# IceTray Services: Options

```
class ExampleModule(icetray.I3Module):
    def __init__(self, context):
        icetray.I3Module.__init__(self, context)
        self.AddParameter("RNG", "I3RandomService", None)

    def Configure(self):
        self.rng = self.GetParameter("RNG")
        if not self.rng:
            self.rng = self.context["I3RandomService"]
            if not self.rng:
                icetray.logging.log_fatal("No RNG found!!!")

    def DAQ(self, frame):
        random_number = self.rng.uniform(math.pi)
        frame["RandomNumber"] = dataclasses.I3Double(random_number)
        self.PushFrame(frame)
```

- 1) As a parameter.
- 2) From the context.

# IceTray Services

## Two Options

- 1) Parameter
- 2) Context

No need for service  
factories anymore

```
class ExampleModule(icetray.I3Module):
    def __init__(self, context):
        icetray.I3Module.__init__(self, context)
        self.AddParameter("RNG", "I3RandomService", None)

    def Configure(self):
        self.rng = self.GetParameter("RNG")
        if not self.rng:
            self.rng = self.context["I3RandomService"]
            if not self.rng:
                icetray.logging.log_fatal("No RNG found!!!")

    def DAQ(self, frame):
        random_number = self.rng.uniform(math.pi)
        frame["RandomNumber"] = dataclasses.I3Double(random_number)
        self.PushFrame(frame)
```

```
tray = I3Tray()
tray.context['I3RandomService'] = phys_services.I3GSLRandomService(42)
tray.Add(GeneratorSegment)
tray.Add(ExampleModule)
tray.Add('Dump')

tray.Execute(10)
```

**NOTE 1:** You still might see "service factories" in production scripts.

Old, complicated way to install a service in a context.



# IceTray Services

## Two Options

- 1) Parameter
- 2) Context

No need for service  
factories anymore

```
class ExampleModule(icetray.I3Module):
    def __init__(self, context):
        icetray.I3Module.__init__(self, context)
        self.AddParameter("RNG", "I3RandomService", None)

    def Configure(self):
        self.rng = self.GetParameter("RNG")
        if not self.rng:
            self.rng = self.context["I3RandomService"]
            if not self.rng:
                icetray.logging.log_fatal("No RNG found!!!")

    def DAQ(self, frame):
        random_number = self.rng.uniform(math.pi)
        frame["RandomNumber"] = dataclasses.I3Double(random_number)
        self.PushFrame(frame)
```

```
tray = I3Tray()
tray.context['I3RandomService'] = phys_services.I3GSLRandomService(42)
tray.Add(GeneratorSegment)
tray.Add(ExampleModule)
tray.Add('Dump')

tray.Execute(10)
```

**NOTE 2:** When writing post-modern classic I3Modules, DO NOT forget to "push the frame." Failure to push the frame effectively filters it from the stream.

# Services

## Two Options

- 1) Parameter
- 2) Context

```
class ExampleModule(icetray.I3Module):
    def __init__(self, context):
        icetray.I3Module.__init__(self, context)
        self.AddParameter("RNG", "I3RandomService", None)

    def Configure(self):
        self.rng = self.GetParameter("RNG")
        if not self.rng:
            self.rng = self.context["I3RandomService"]
            if not self.rng:
                icetray.logging.log_fatal("No RNG found!!!")

    def DAQ(self, frame):
        random_number = self.rng.uniform(math.pi)
        frame["RandomNumber"] = dataclasses.I3Double(random_number)
        self.PushFrame(frame)
```

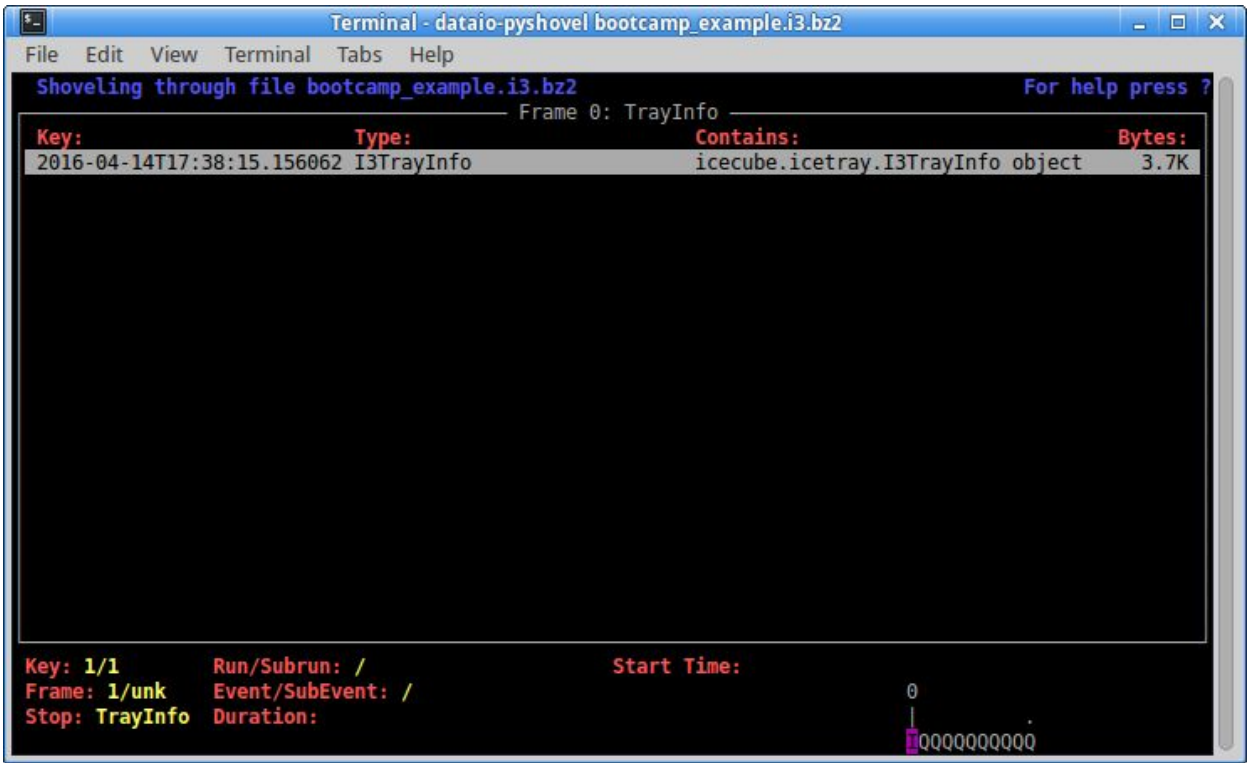
Generate 10 frames.

Tack on an I3Writer  
to generate an I3File.

```
tray = I3Tray()
tray.context["I3RandomService"] = phys_services.I3GSLRandomService(42)
tray.Add(GeneratorSegment)
tray.Add(ExampleModule)
tray.Add("Dump")
tray.Add("I3Writer", filename = "bootcamp_example.i3.bz2")
tray.Execute(10)
```

# dataio-pyshovel Example

\$ dataio-pyshovel bootcamp\_example.i3.bz2



# Exercises

## In Madison:

cp /home/olivas/bootcamp.py .

## Outside Madison:

scp -S ssh <username>@data.icecube.wisc.edu:/home/olivas/bootcamp.py .

- 1) Add an I3Particle neutrino primary to the tree.
- 2) Add an I3Particle muon as a secondary.
- 3) Randomize the muon energy.
- 4) Change the filter to only pass frames with muon energy above 300 GeV
- 5) Implement a DAQ for ExampleModule method that prints the tree.

```
#!/usr/bin/env python
from I3Tray import I3Tray
from icecube import icetray, dataio, dataclasses

@icetray.traysegment
def GeneratorSegment(tray, name):
    def generator(frame):
        frame["tree"] = dataclasses.I3MCTree()

    tray.Add("I3InfiniteSource")
    tray.Add(generator, streams = [icetray.I3Frame.DAQ])
    tray.Add(lambda frame : frame.Has("tree"),
             streams = [icetray.I3Frame.DAQ])

class ExampleModule(icetray.I3Module):
    def __init__(self, context):
        icetray.I3Module.__init__(self, context)

tray = I3Tray()
tray.Add(GeneratorSegment)
tray.Add(ExampleModule)
tray.Add("Dump")
tray.Execute(10)
```