Intro to IceTray
Madison Bootcamp 2018

Alex Olivas

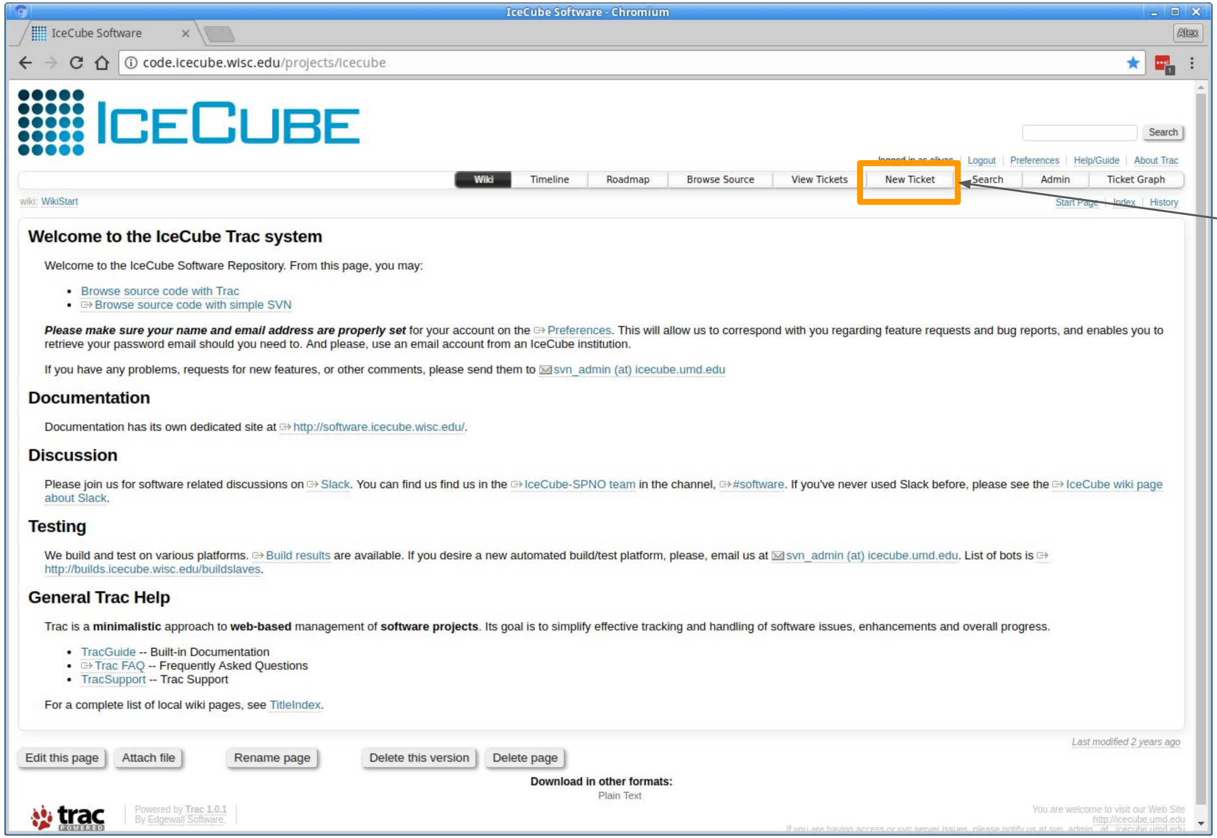
IceTray : The Definitive Guide

Documentation is built nightly:
http://software.icecube.wisc.edu/documentation/projects/icetray/index.html

If you see something, say something.
http://code.icecube.wisc.edu/projects/icecube/newticket

Please, please, please file a ticket if you find issues with IceTray documentation.
We can't fix problems we don't know about.

The Ticketing System http://code.icecube.wisc.edu


IceCube Software - Chromium
IceCube Software
Alex
code.icecube.wisc.edu/projects/icecube
ICECUBE
Search
logged in as olivas | Logout | Preferences | Help/Guide | About Trac
Wiki | Timeline | Roadmap | Browse Source | View Tickets | New Ticket | Search | Admin | Ticket Graph
wiki: WikiStart
Start Page | Index | History

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

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. 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

Download in other formats:
Plain Text

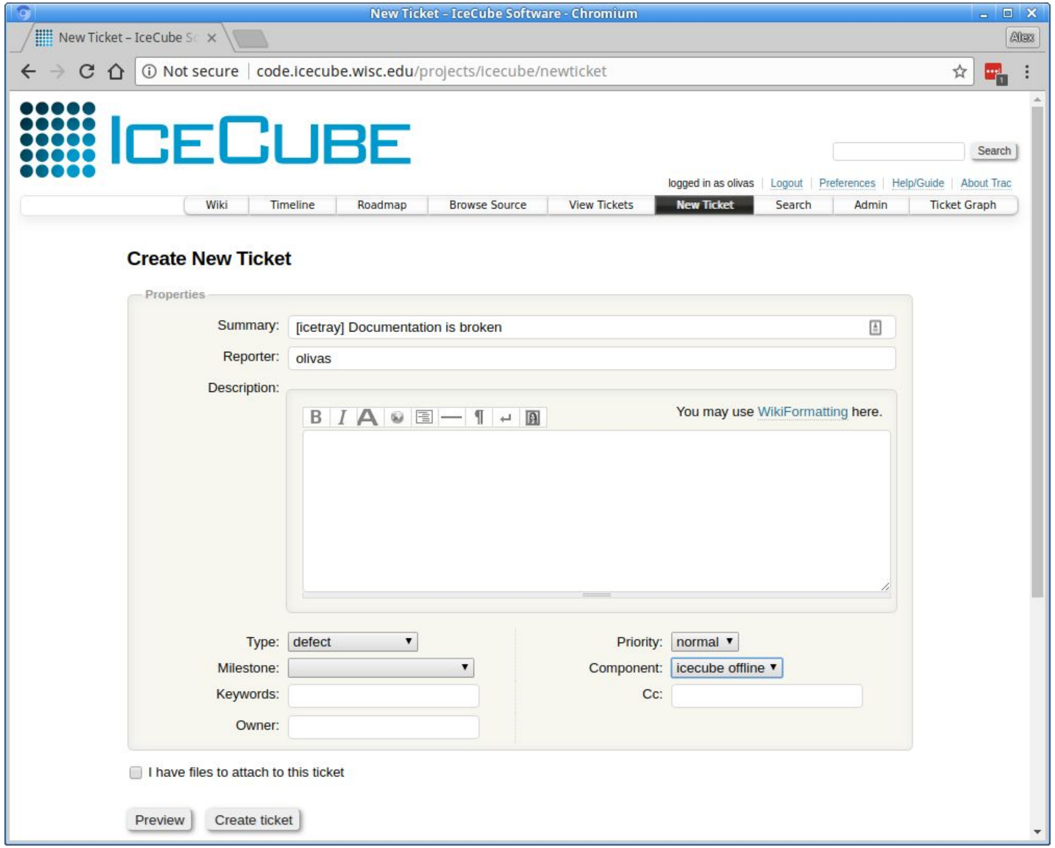
Powered by Trac 1.0.1
By Edgewall Software.

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.

The Ticketing System http://code.icecube.wisc.edu


New Ticket - IceCube Software - Chromium

New Ticket - IceCube S

Not secure | code.icecube.wisc.edu/projects/icecube/newticket

logged in as olivas | Logout | Preferences | Help/Guide | About Trac

Wiki | Timeline | Roadmap | Browse Source | View Tickets | New Ticket | Search | Admin | Ticket Graph

Create New Ticket

Properties

Summary:	[icetray] Documentation is broken		
Reporter:	olivas		
Description:	You may use WikiFormatting here.		
Type:	defect	Priority:	normal
Milestone:		Component:	icecube offline
Keywords:		Cc:	
Owner:			

☐ I have files to attach to this ticket

Preview | Create ticket

One and only one hard and fast rule:

Do not submit tickets as 'icecube'

Fill in the fields as best you can.

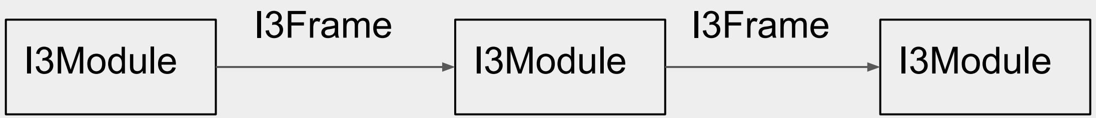
Convention: In the description, it's helpful, but not necessary, to start with the project name in square brackets.

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.

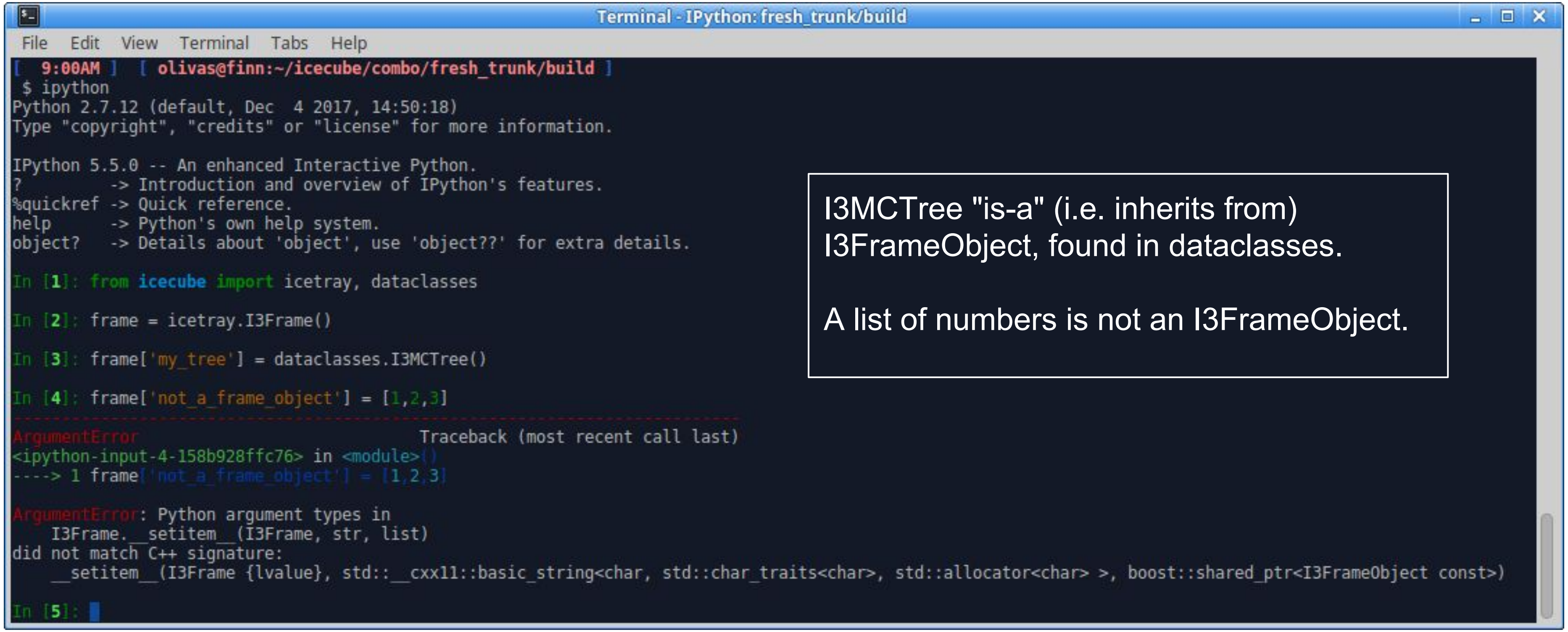
*Trivia : I3Frame is actually a C++ class which manages map<string, shared_ptr<I3FrameObject>>

$$\begin{tikzcd}\boxed{\text{I3Module}} \arrow[r, "\text{I3Frame}"] & \boxed{\text{I3Module}} \arrow[r, "\text{I3Frame}"] & \boxed{\text{I3Module}}\end{tikzcd}$$

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

- dataclasses - http://software.icecube.wisc.edu/documentation/projects/dataclasses/index.html
- simclasses - http://software.icecube.wisc.edu/documentation/projects/simclasses/index.html
- recclasses - http://software.icecube.wisc.edu/documentation/projects/recclasses/index.html

IceTray : I3FrameObjects

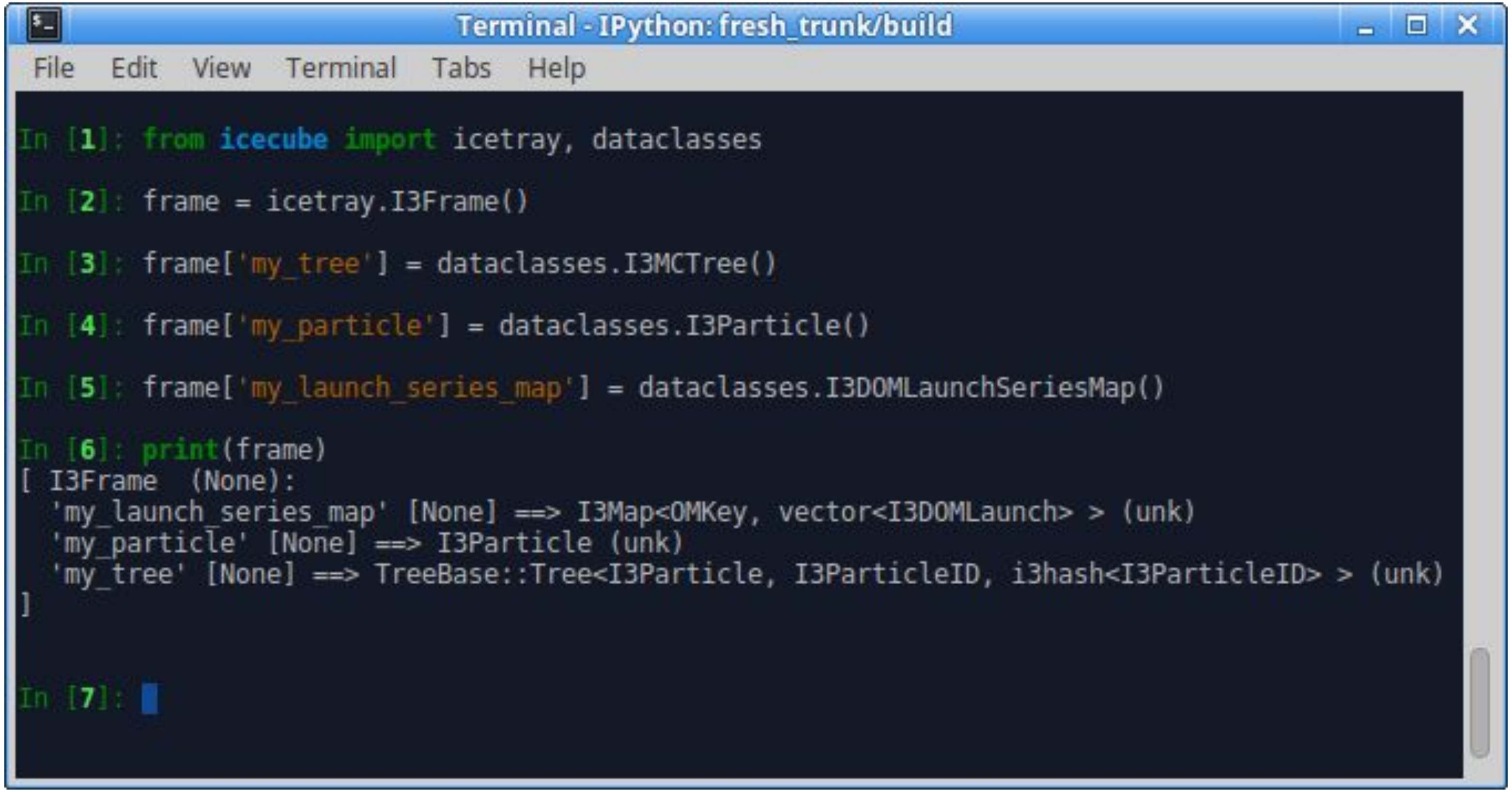

Terminal - IPython: fresh_trunk/build
File Edit View Terminal Tabs Help

```
[  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.

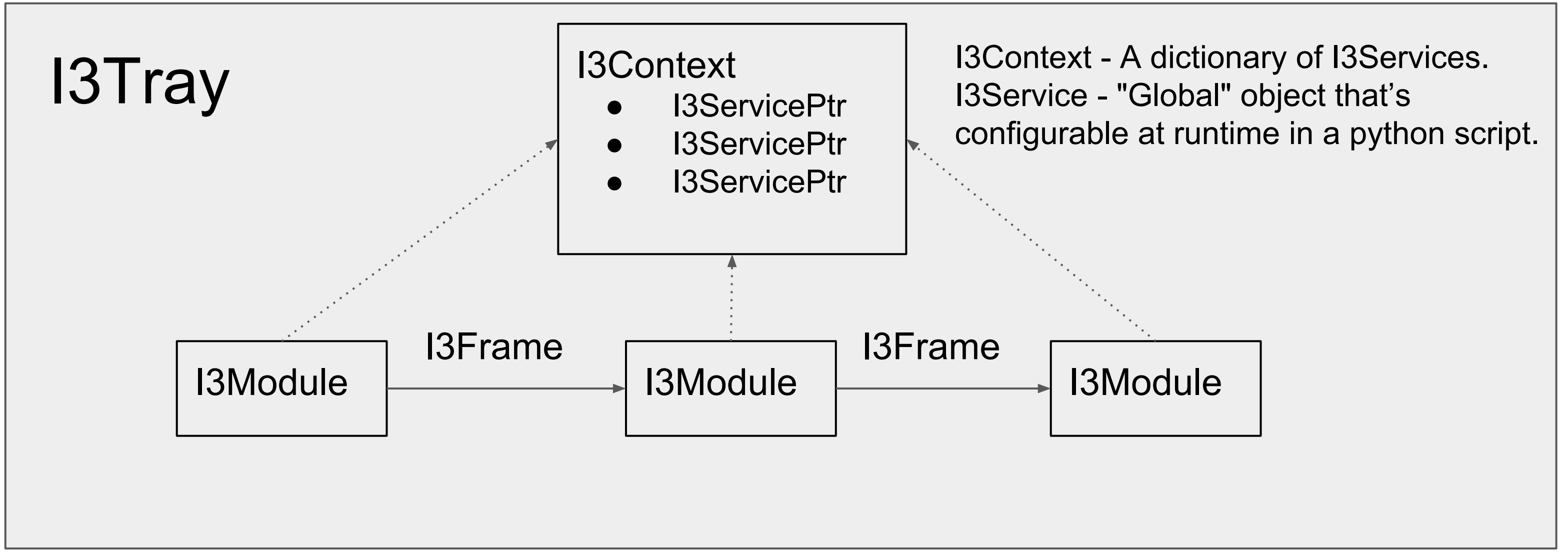
A list of numbers is not an I3FrameObject.

IceTray : I3FrameObjects


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]: 
```

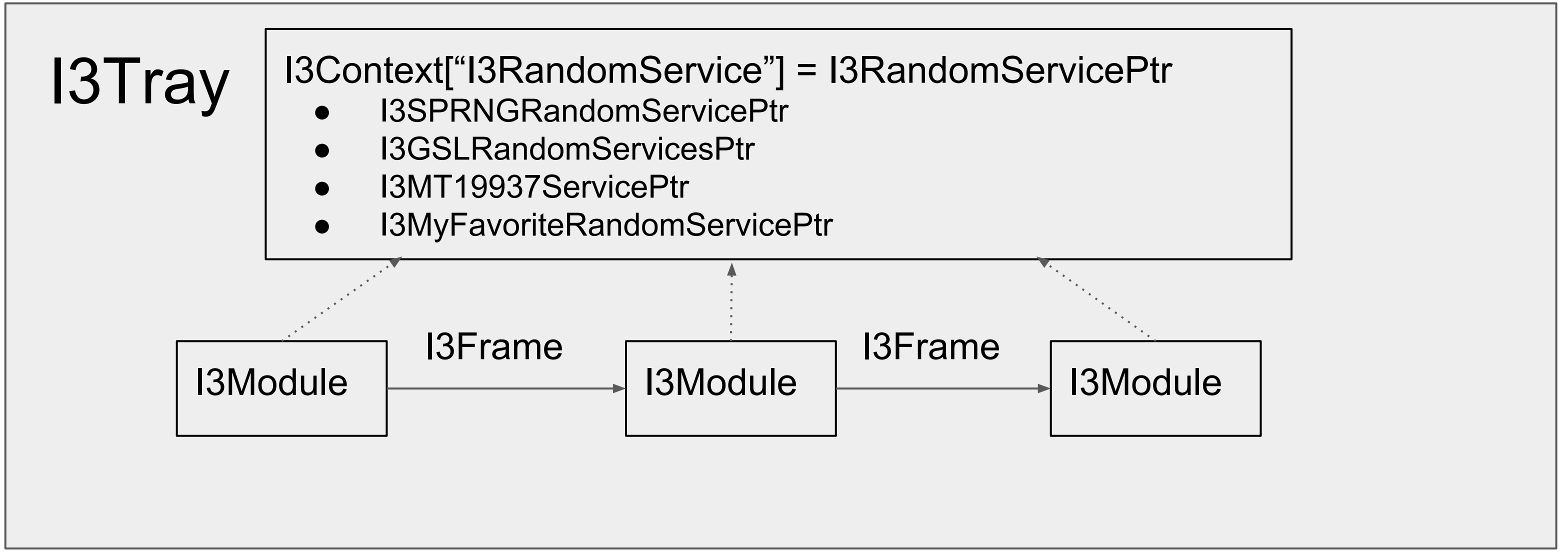
IceTray : I3Context and I3Services


I3Tray
I3Context
● I3ServicePtr
● I3ServicePtr
● I3ServicePtr
I3Context - A dictionary of I3Services.
I3Service - "Global" object that's configurable at runtime in a python script.
I3Module
I3Frame
I3Module
I3Frame
I3Module

When to use a proper I3Service stored in the I3Context?

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

IceTray : RNG Service Example


I3Tray
I3Context[“I3RandomService”] = I3RandomServicePtr
● I3SPRNGRandomServicePtr
● I3GSLRandomServicesPtr
● I3MT19937ServicePtr
● I3MyFavoriteRandomServicePtr
I3Module
I3Frame
I3Module
I3Frame
I3Module

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

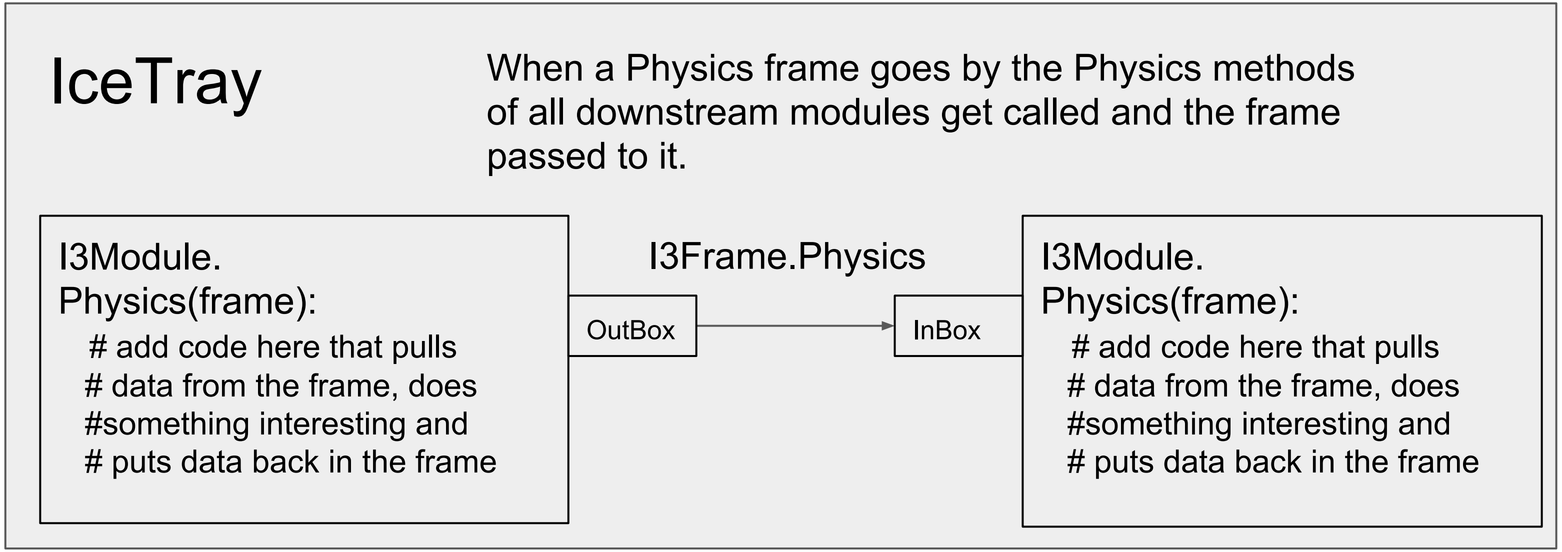
IceTray : The Frame-Stream-Stop Model

Frames come in different flavors

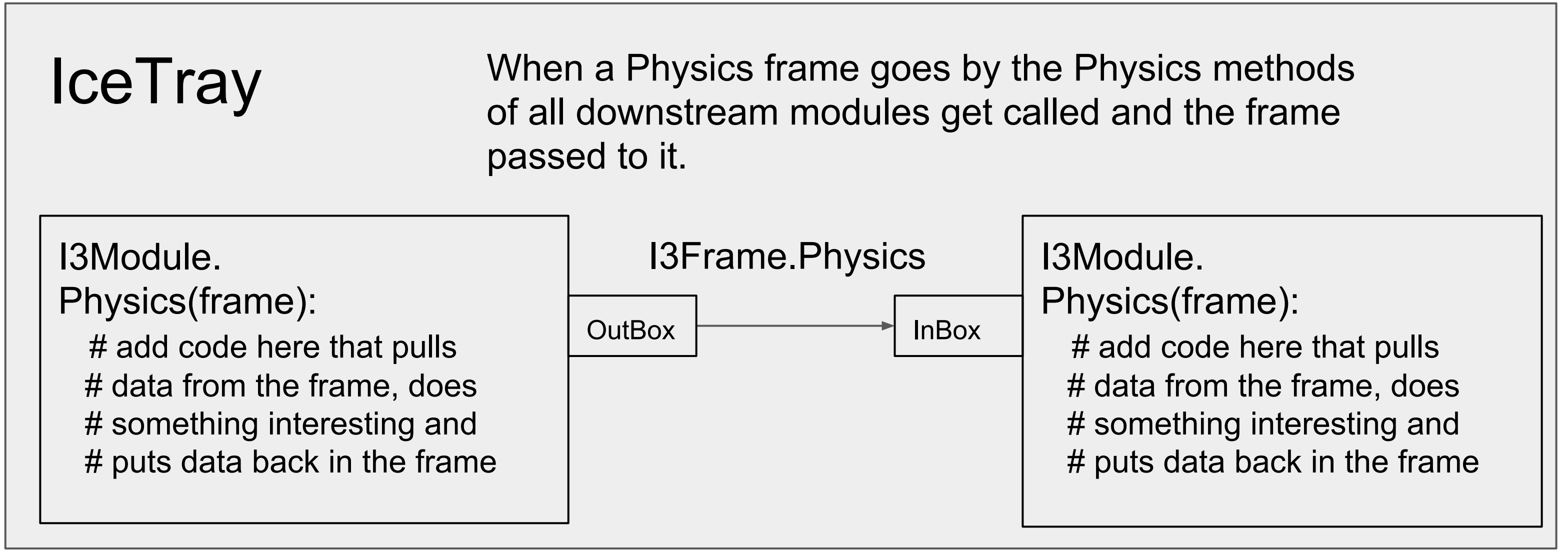
Stops are methods that correspond to a frame type.

I3Frame Types	I3Module ‘Stops’
I3Frame::TrayInfo	
I3Frame::Geometry	I3Module::Geometry
I3Frame::Calibration	I3Module::Calibration
I3Frame::DetectorStatus	I3Module::DetectorStatus
I3Frame::Physics	I3Module::Physics
I3Frame::DAQ	I3Module::DAQ

IceTray : The Frame-Stream-Stop Model

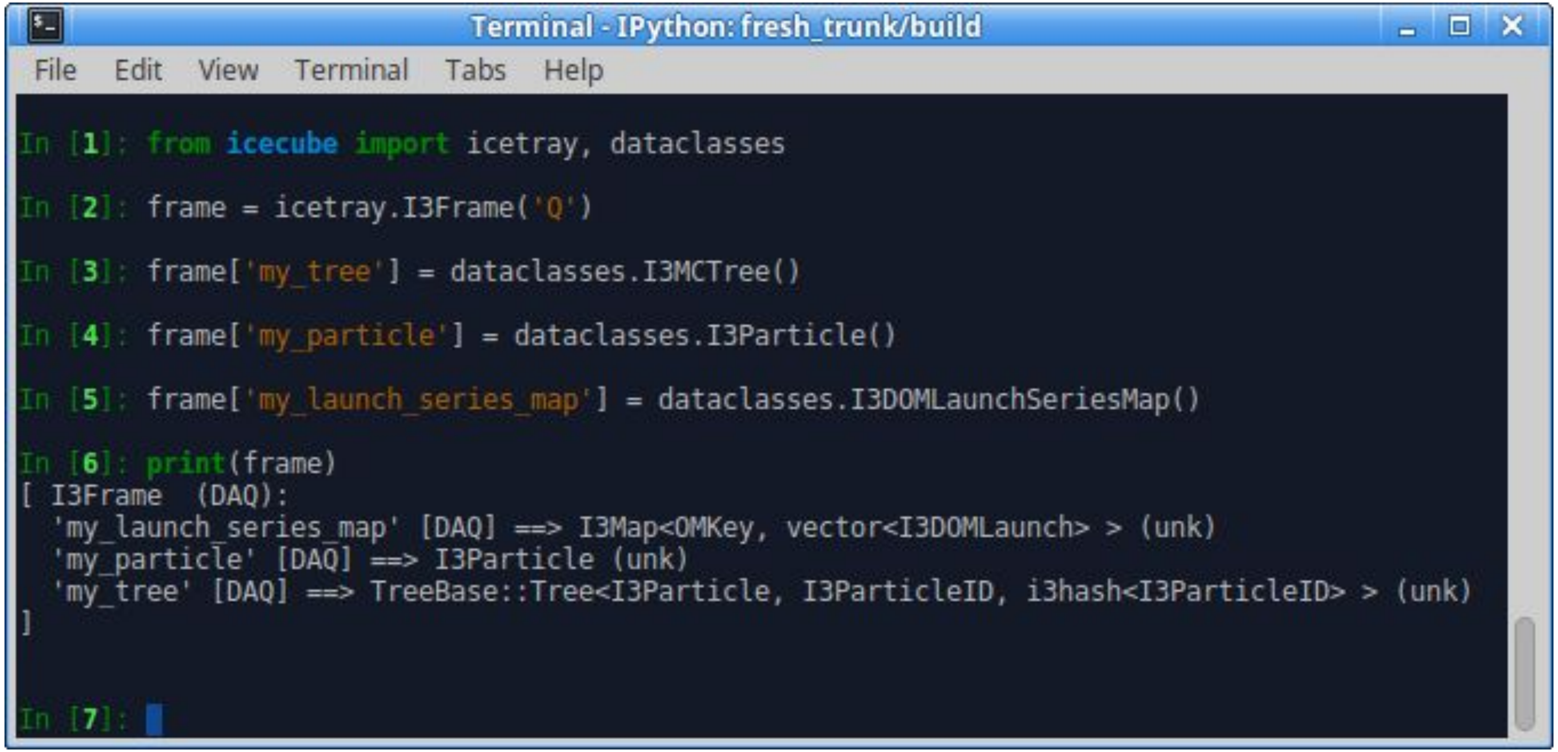

IceTray
When a Physics frame goes by the Physics methods of all downstream modules get called and the frame passed to it.
I3Module.
Physics(frame):
# add code here that pulls
# data from the frame, does
#something interesting and
# puts data back in the frame
OutBox
I3Frame.Physics
InBox
I3Module.
Physics(frame):
# add code here that pulls
# data from the frame, does
#something interesting and
# puts data back in the frame

IceTray : The Frame-Stream-Stop Model


IceTray
When a Physics frame goes by the Physics methods of all downstream modules get called and the frame passed to it.
I3Module.
Physics(frame):
# add code here that pulls
# data from the frame, does
# something interesting and
# puts data back in the frame
OutBox
I3Frame.Physics
InBox
I3Module.
Physics(frame):
# add code here that pulls
# data from the frame, does
# something interesting and
# puts data back in the frame

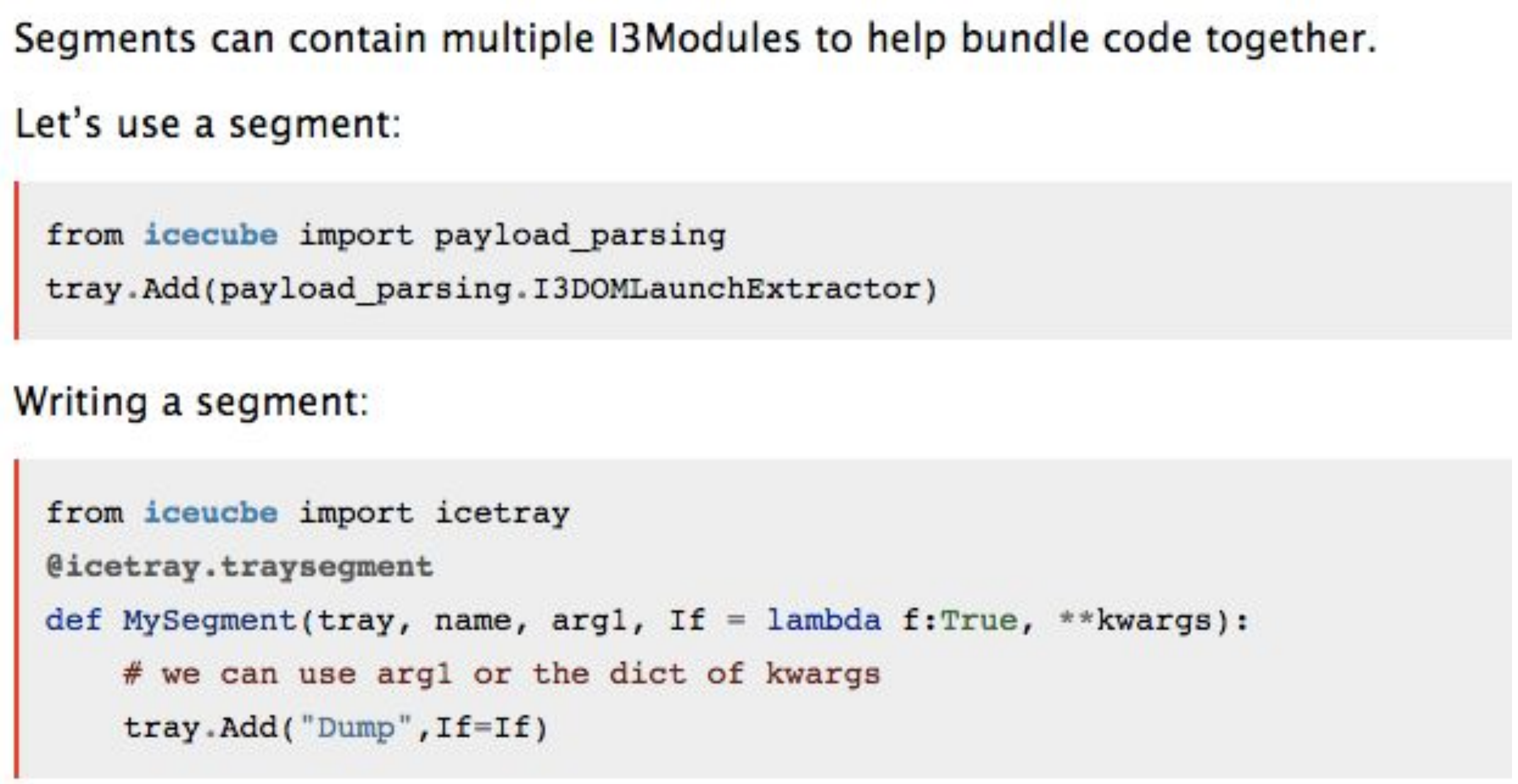
Frame Hierarchy - Generally expected in this order: **GCDQP**
Frame Mixing - Example: All objects from G, C, D, Q are accessible from P-frames.
Frame Packets - **QPPPPP** P-frames following Q-frame "belong" to the Q-frame.
I3PacketModule - Allows you to process a vector of frames (i.e. "packet").

IceTray : 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 : 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 iceucbe 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)
```

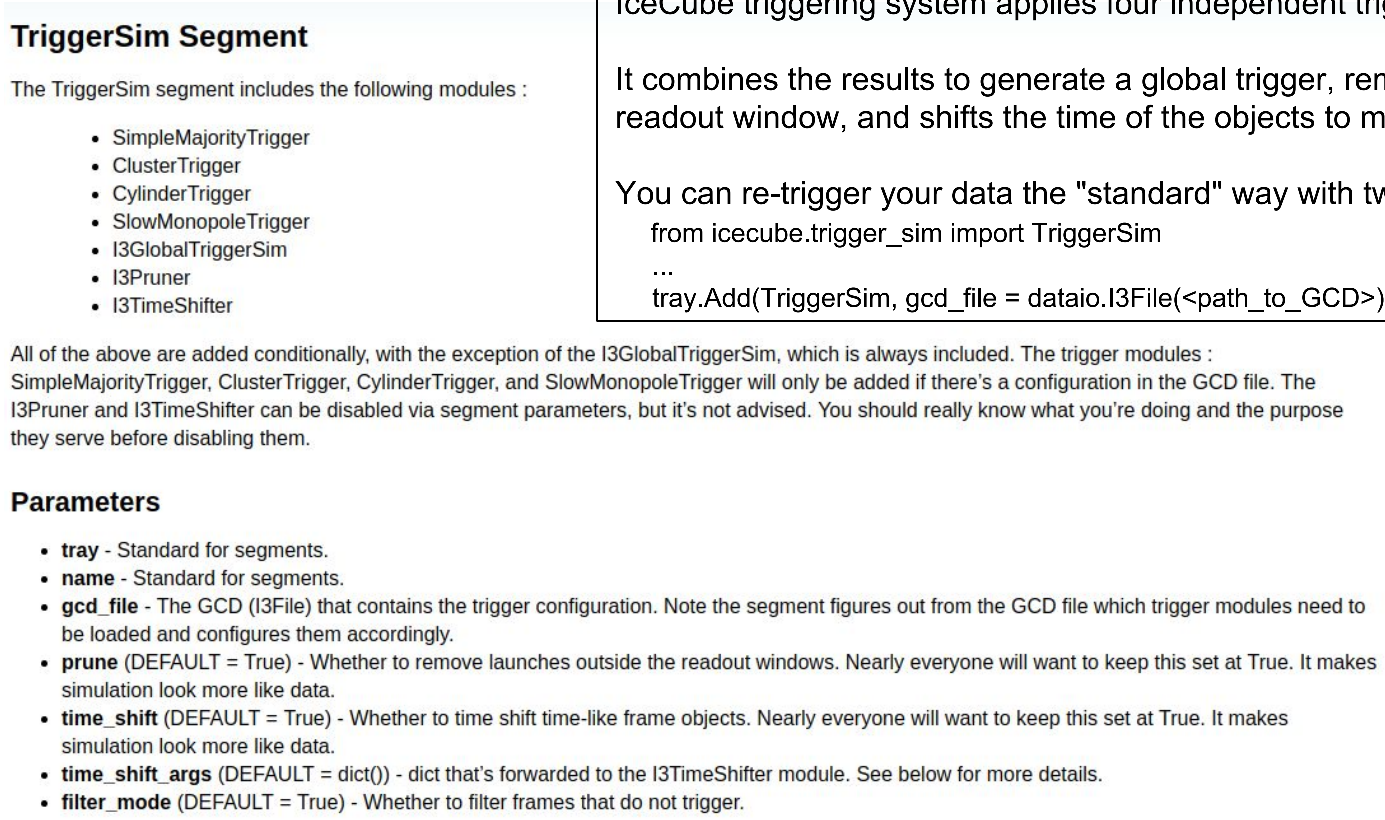
TriggerSim Segment

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>)
```



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.

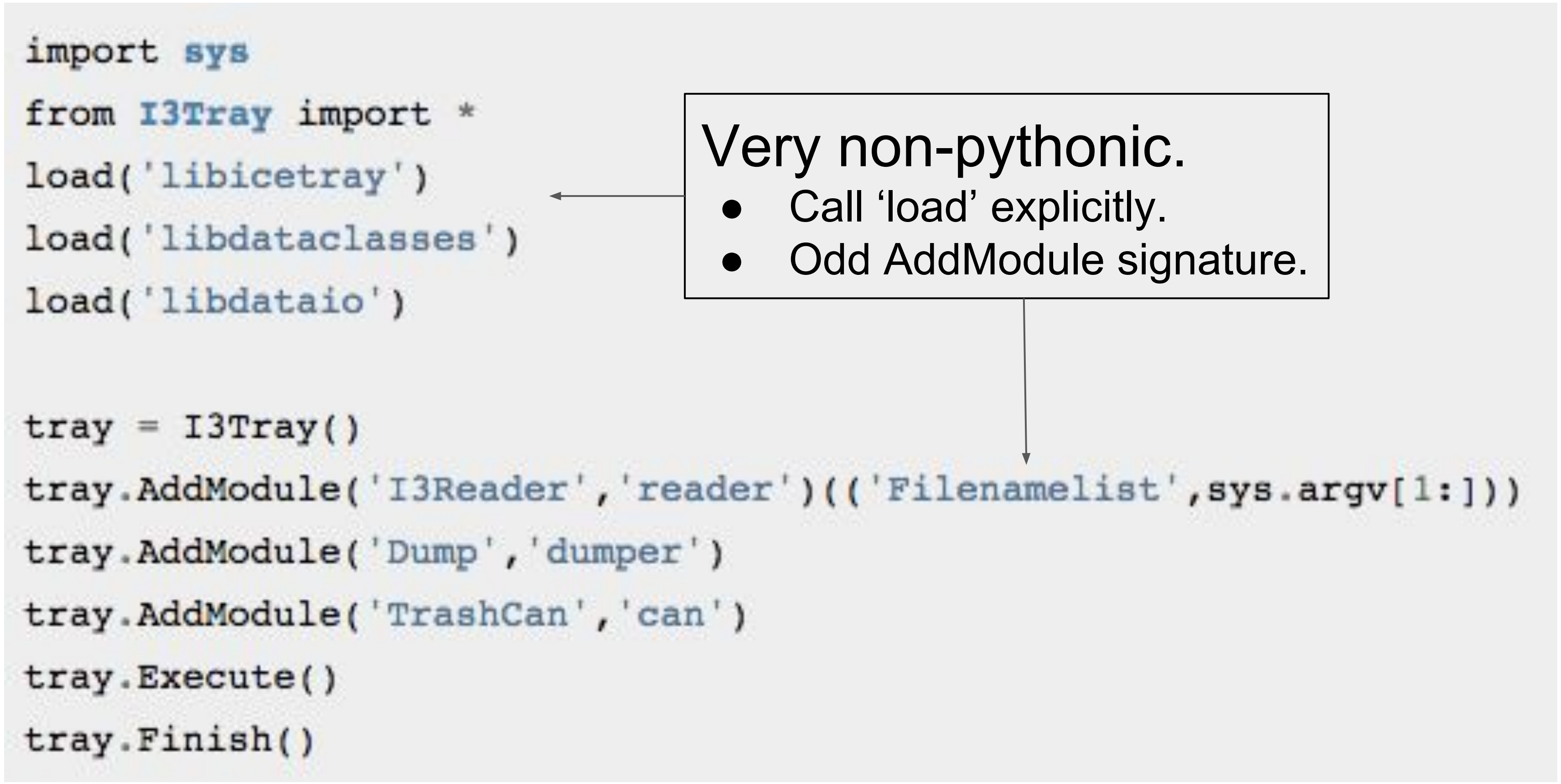
IceTray History Lesson

From 2008 - Present...

Why? Because lots of production scripts haven't been updated in more than 10 years.

IceTray : Pre-2008

*Still lots of production scripts and segments that haven't been cleaned up in over 10 years.

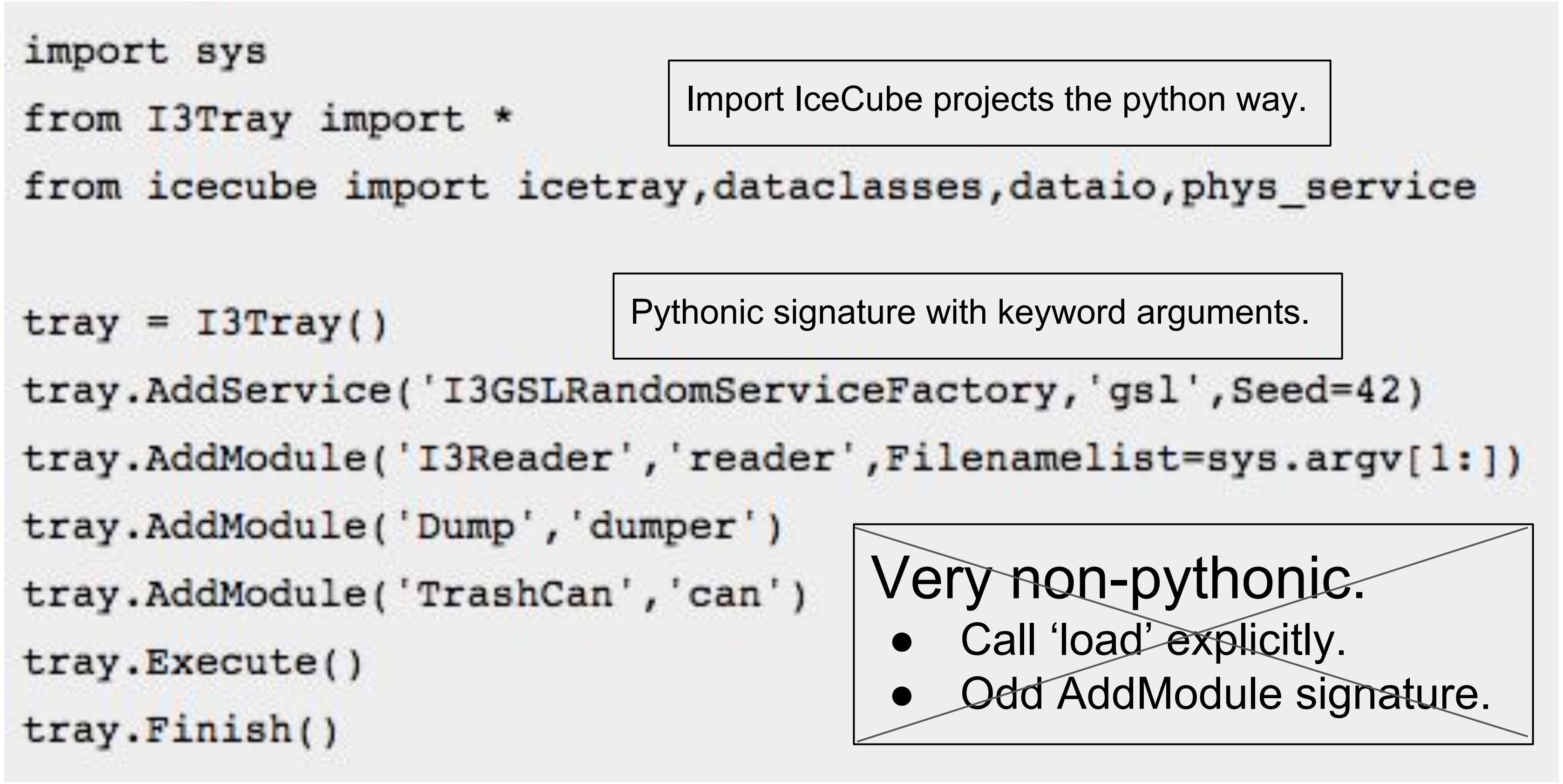


```
import sys
from I3Tray import *
load('libicetray')
load('libdataclasses')
load('libdataio')

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

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()
```

Import IceCube projects the python way.

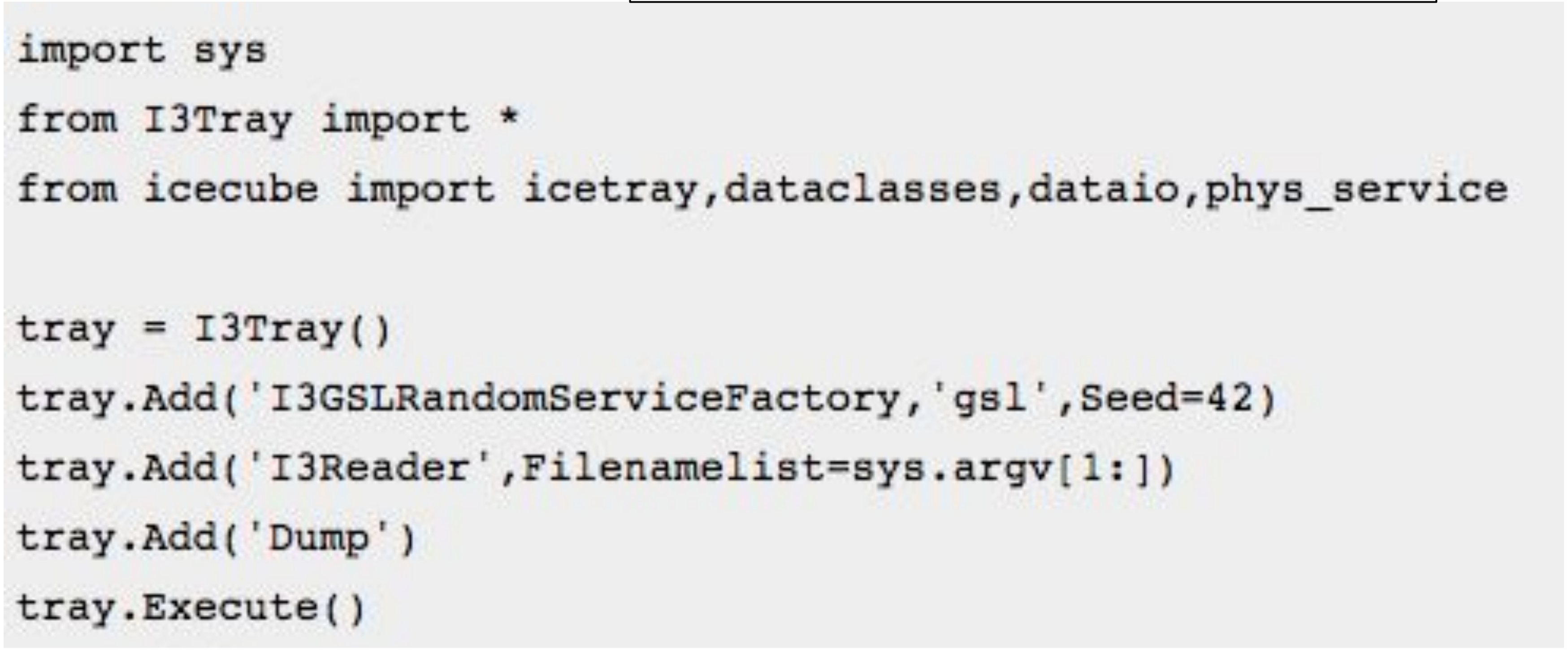
Pythonic signature with keyword arguments.

~~Very non-pythonic.~~

- ~~Call ‘load’ explicitly.~~
- ~~Odd AddModule signature.~~

IceTray : Post-2013

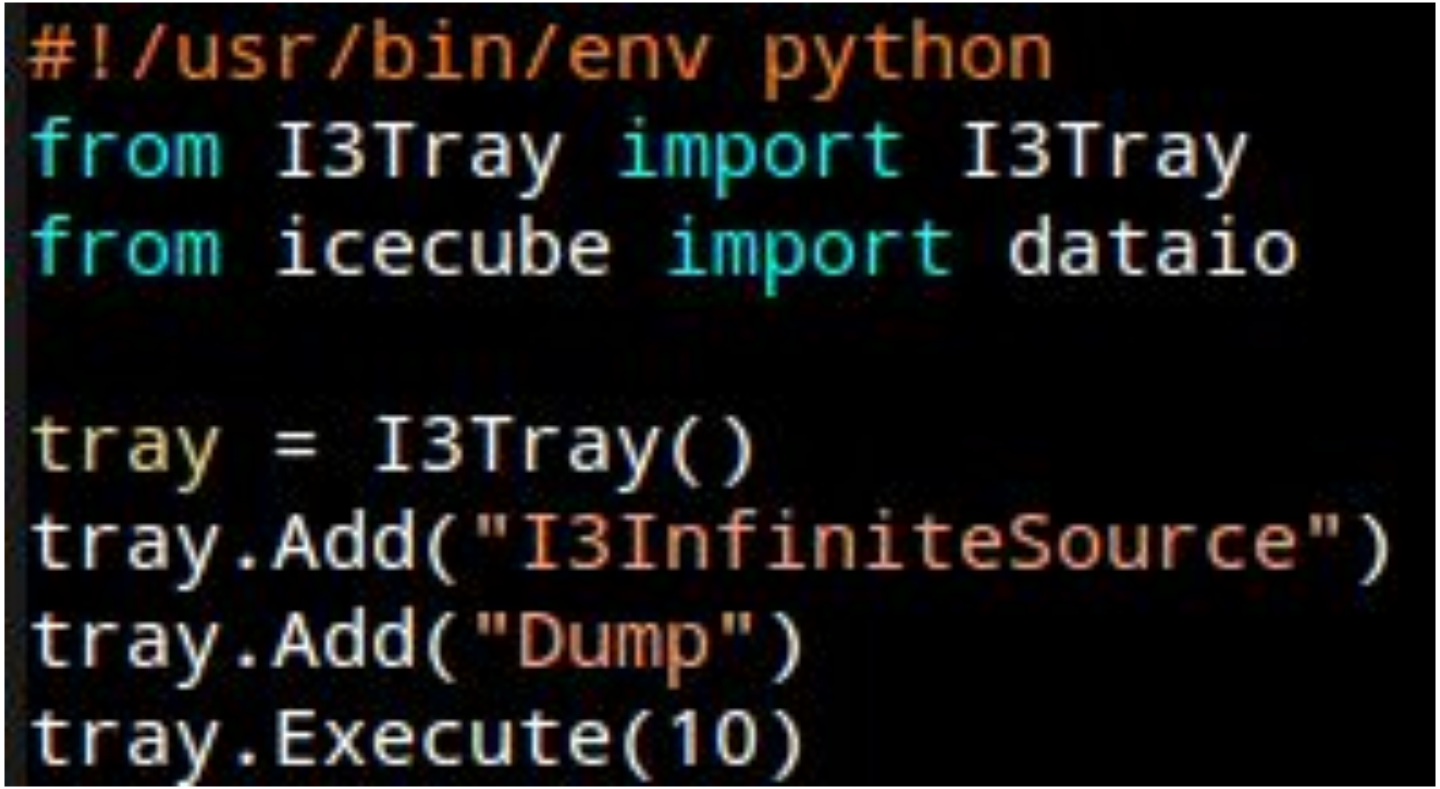
Cleanups
- Just 'Add'
- Anonymous I3Modules - No need to include a name.
- No need to add “TrashCan” Module.
- Need to call “Finish” explicitly.



```
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()
```

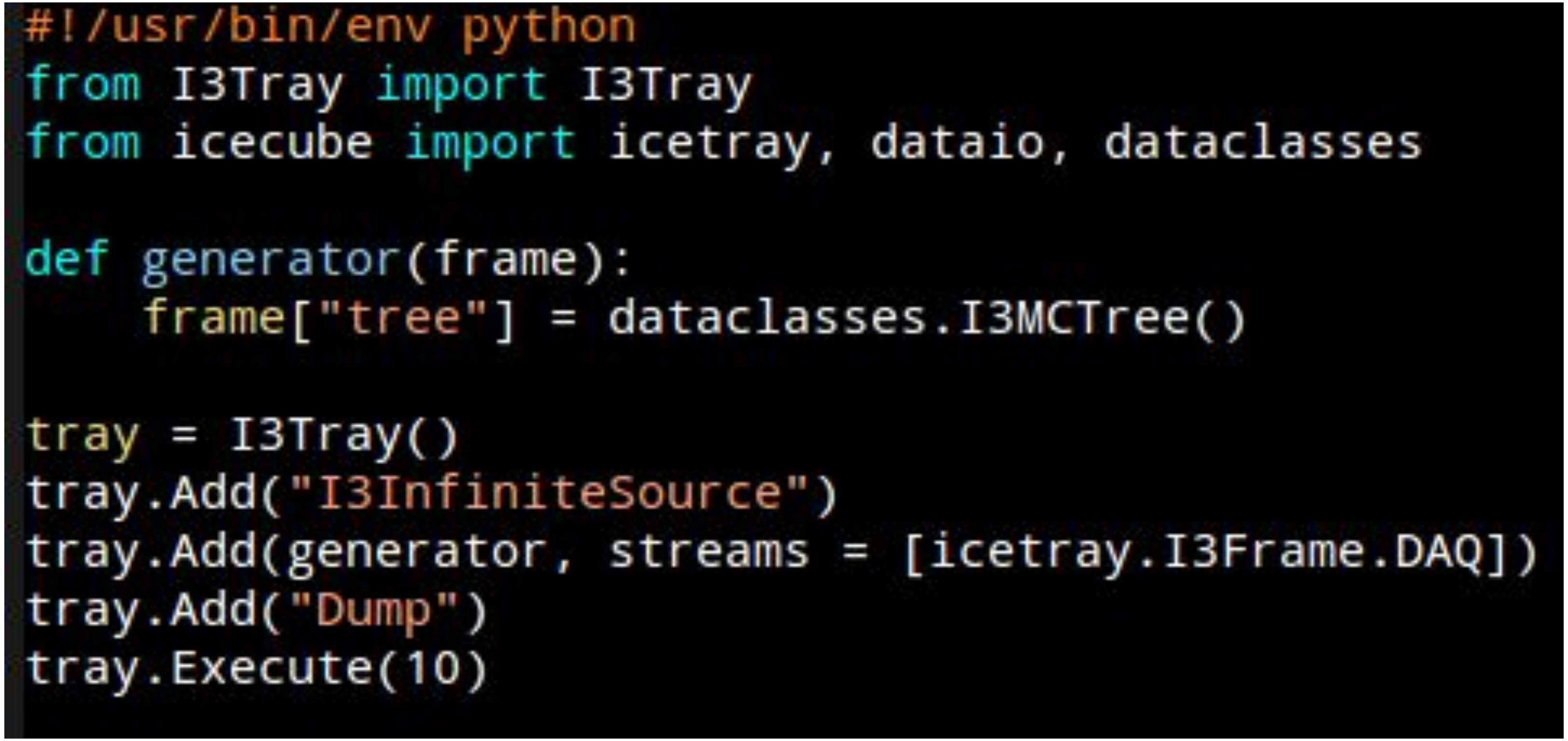
The Simplest IceTray Chain

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)
```

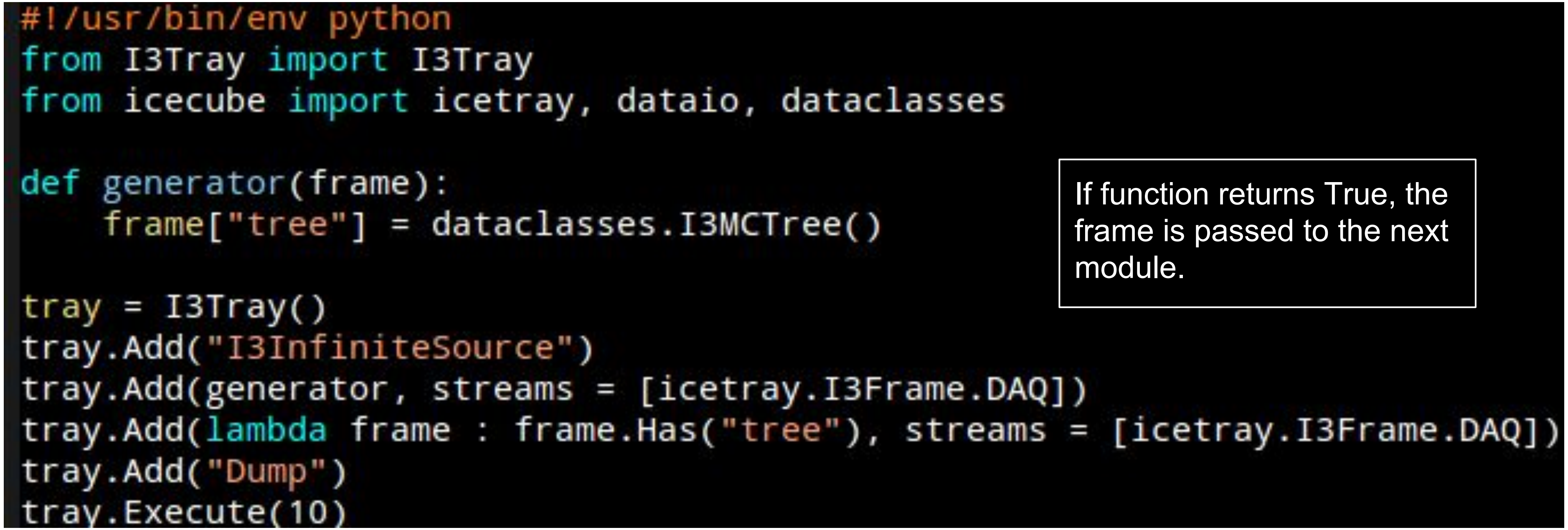
IceTray : Functions as IceTray Modules



```python
#!/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.

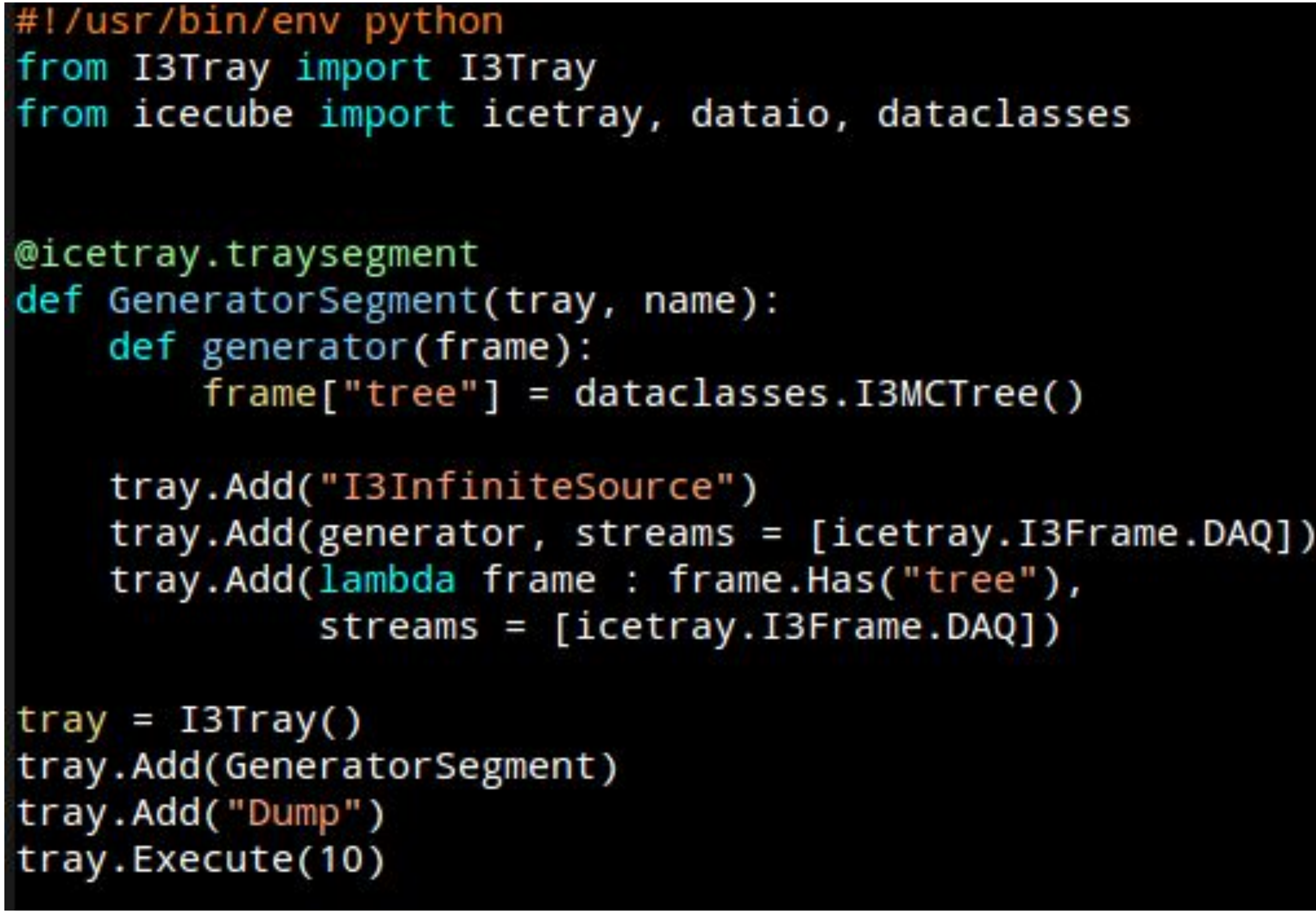


```
#!/usr/bin/env python
from I3Tray import I3Tray
from icecube import icetray, dataio, dataclasses

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.

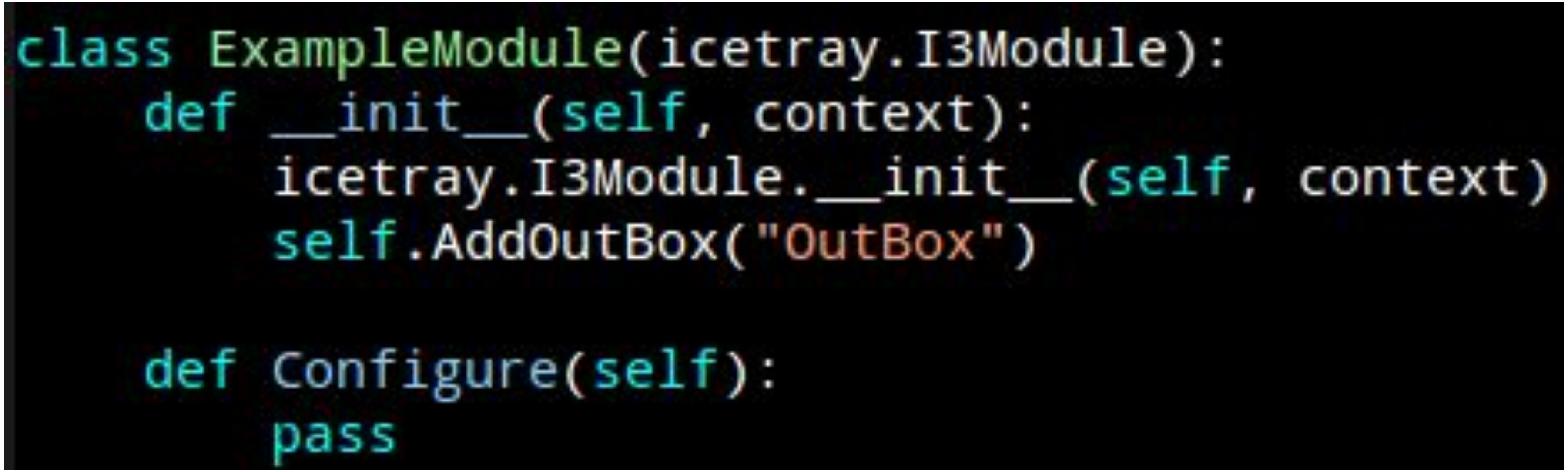
Tray Segments

Group several ‘modules’ and functions together to form something that can plug in to IceTray.



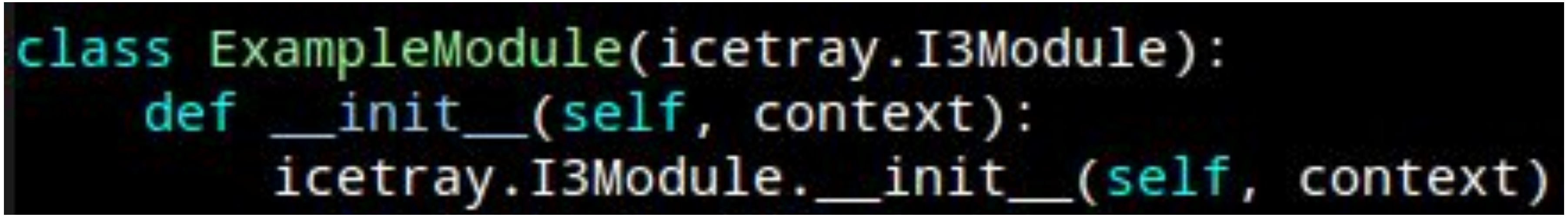
```
#!/usr/bin/env python
from I3Tray import I3Tray
from icecube import icetray, dataio, dataclasses


@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])

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



```python
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.

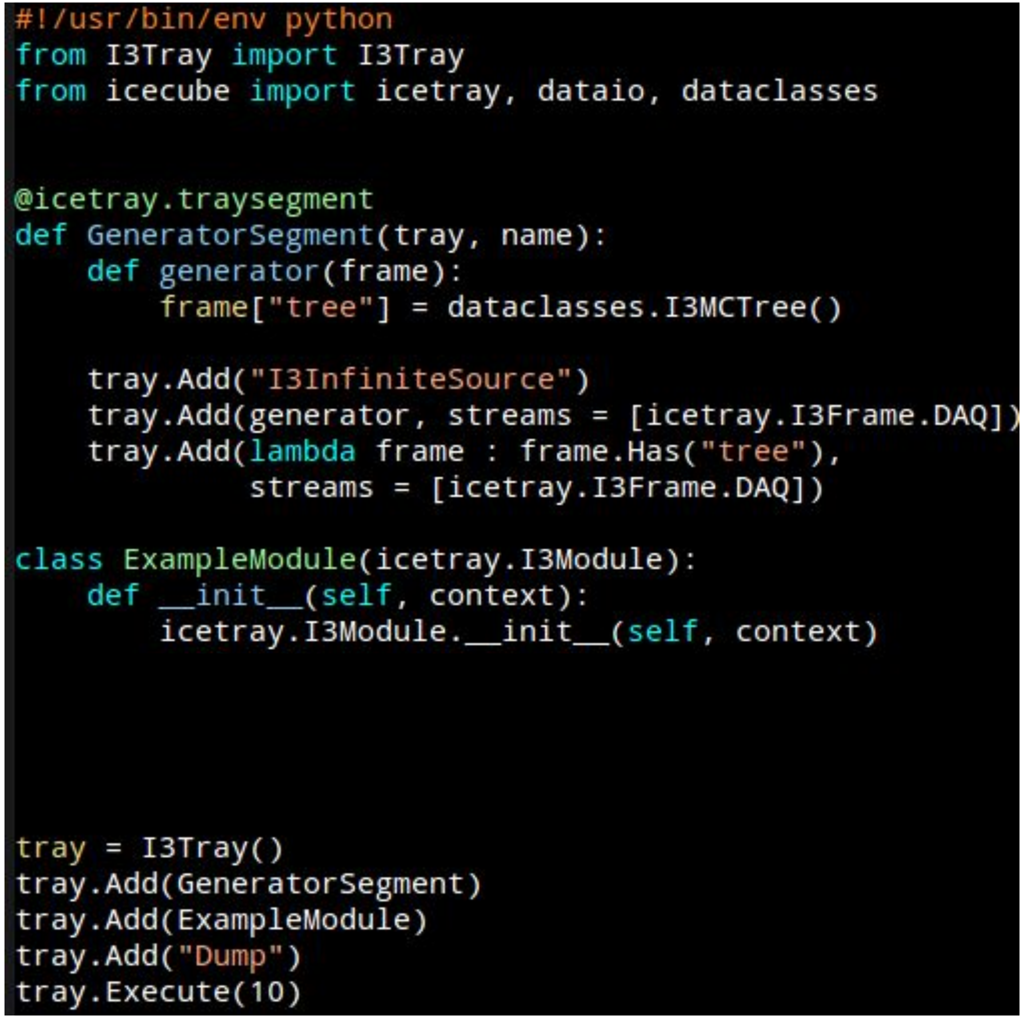
I3Module :

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.

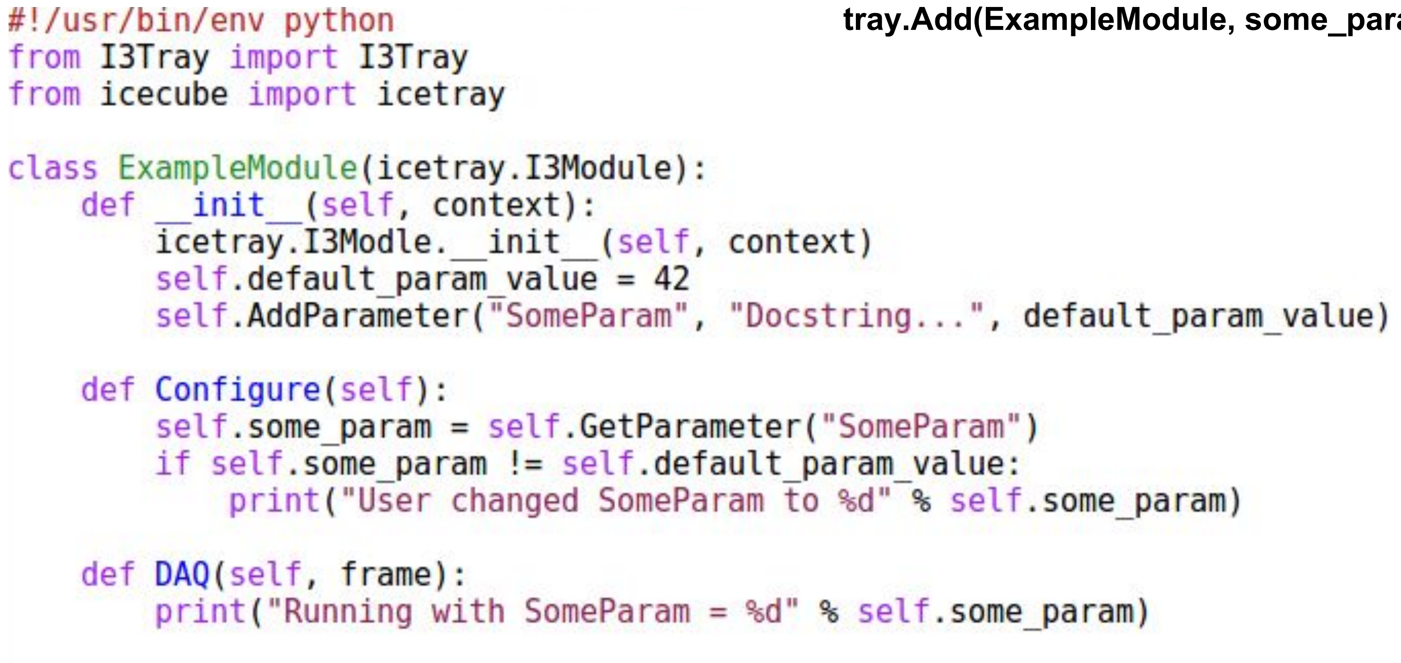


```
#!/usr/bin/env python
from I3Tray import I3Tray
from icecube import icetray, dataio, dataclasses


@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

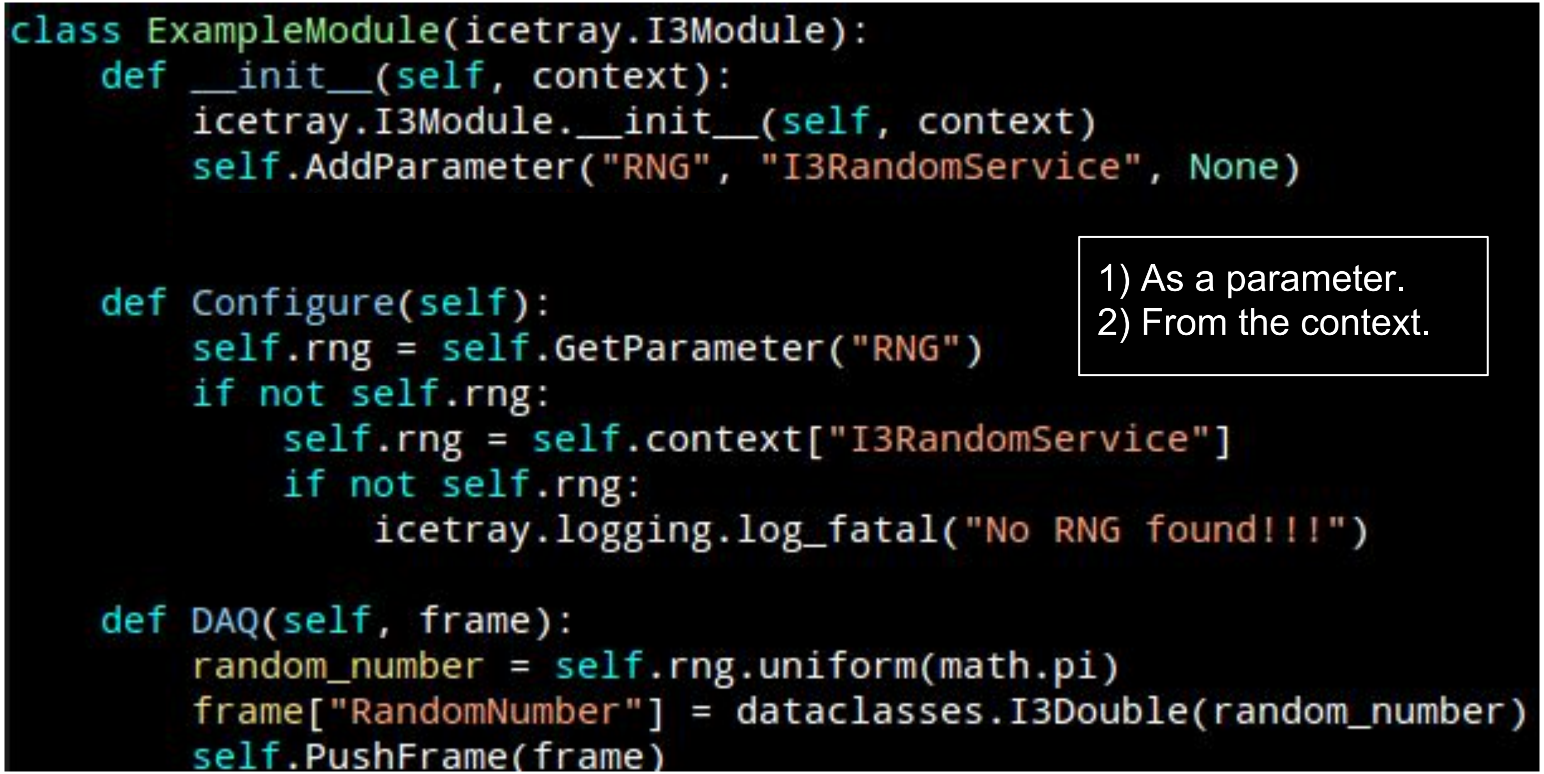
Parameters defined with 'AddParameter' become keyword arguments when added to an I3Tray instance.

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



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

class ExampleModule(icetray.I3Module):
    def __init__(self, context):
        icetray.I3Modle.__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)
```

IceTray Services: Options



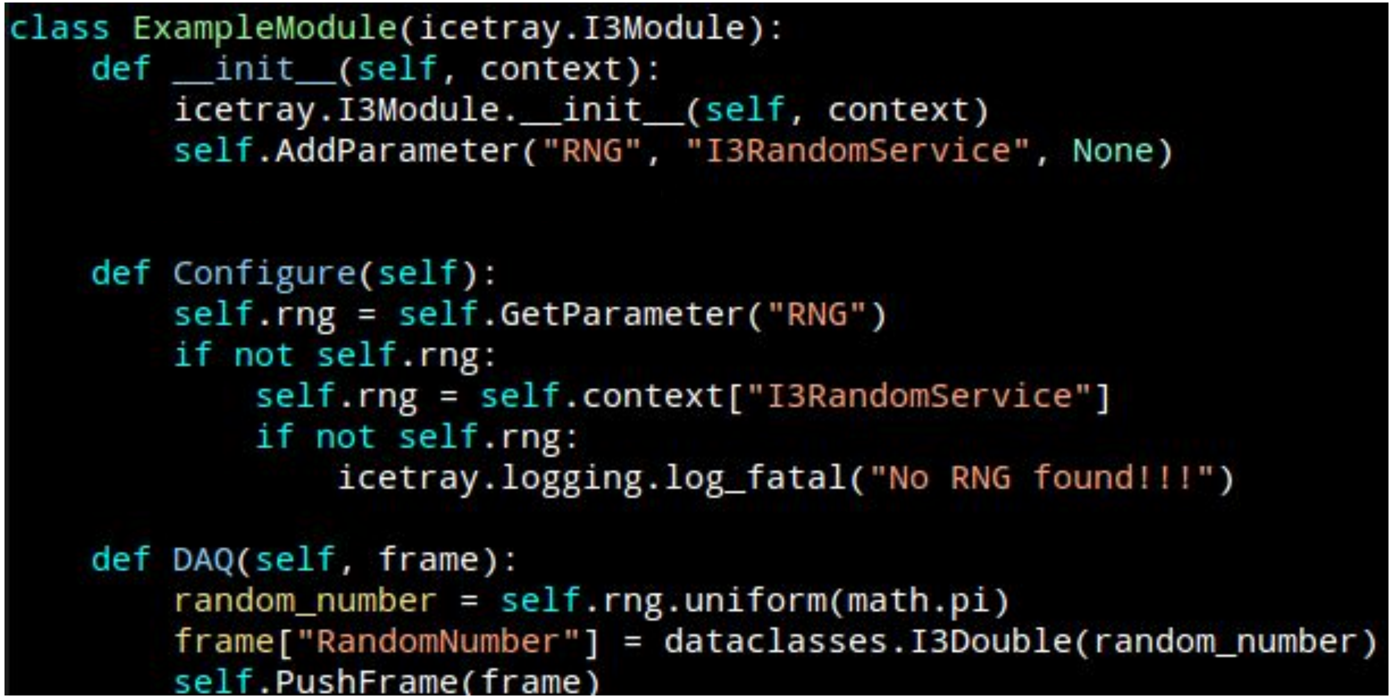
```
class ExampleModule(icetray.I3Module):
    def __init__(self, context):
        icetray.I3Module.__init__(self, context)
        self.AddParameter("RNG", "I3RandomService", None)


    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

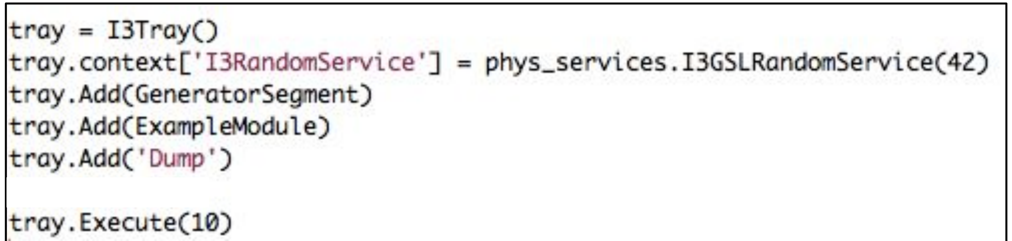
Two Options

1) Parameter
2) Context



```python
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)
```

No need for service factories anymore



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

tray.Execute(10)
```

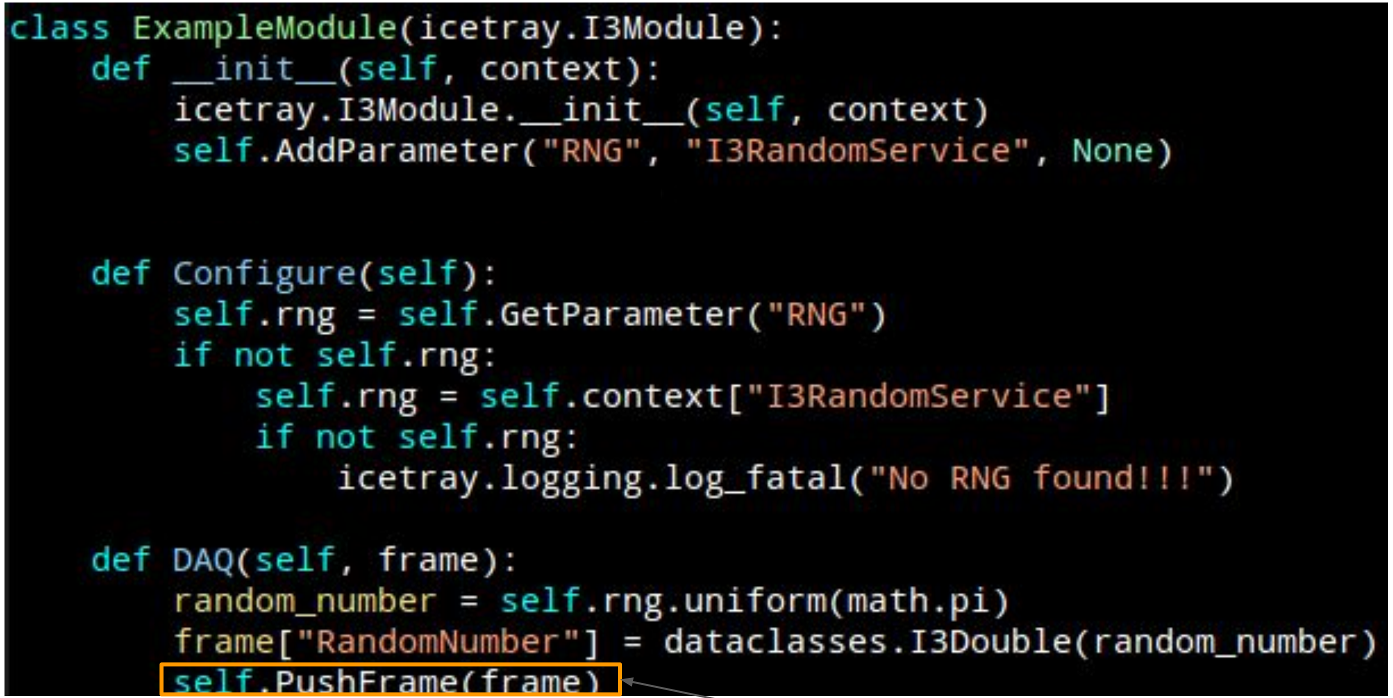
NOTE 1: You still might see "service factories" in production scripts.

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

IceTray Services

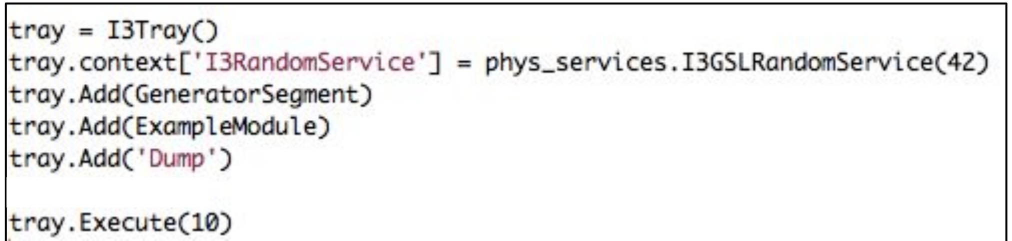
Two Options

1) Parameter
2) Context



```
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)
```

No need for service factories anymore



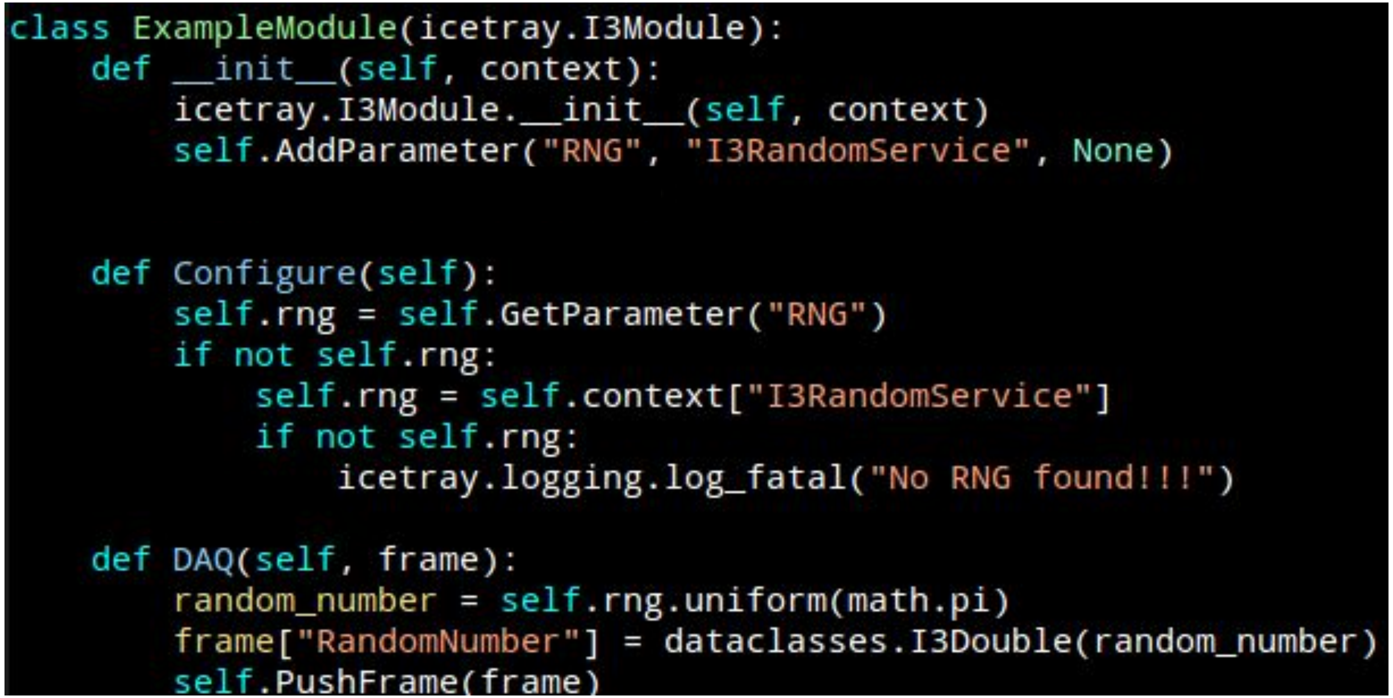
```
tray = I3Tray()
tray.context['I3RandomService'] = phys_services.I3GSLRandomService(42)
tray.Add(GeneratorSegment)
tray.Add(ExampleModule)
tray.Add('Dump')

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

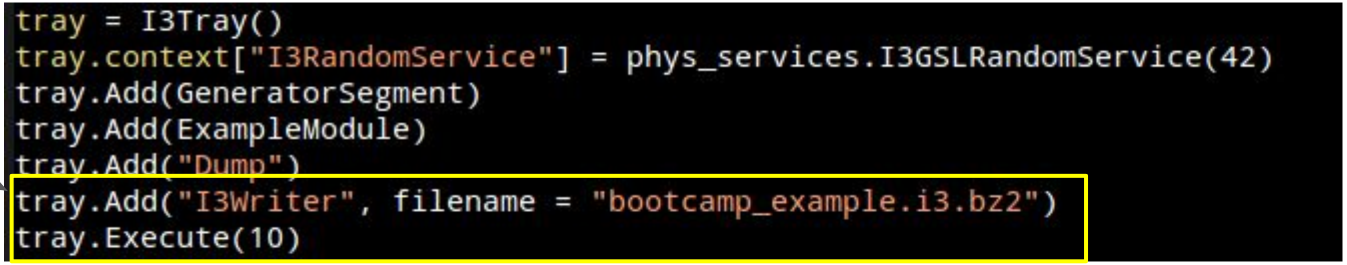
1) Parameter
2) Context



```python
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.

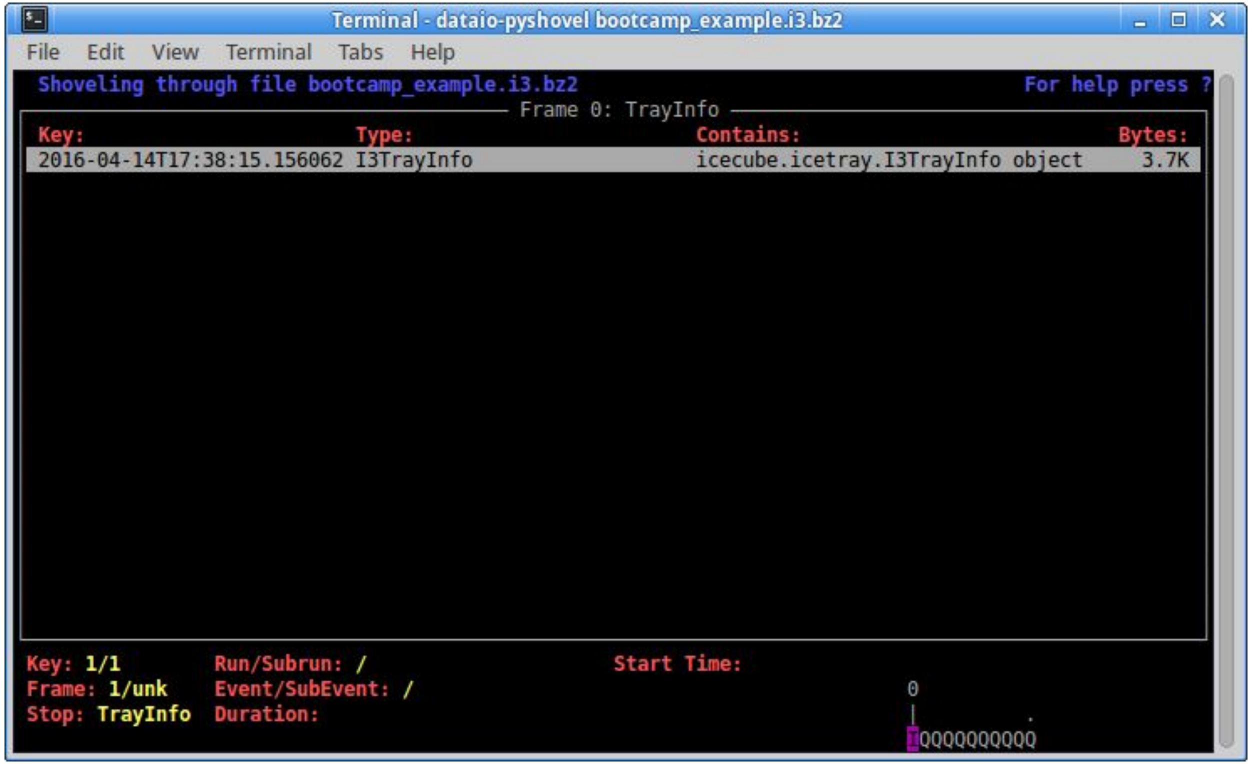
Tack on an I3Writer to generate an I3File.



```python
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


Terminal - dataio-pyshovel bootcamp_example.i3.bz2
File Edit View Terminal Tabs Help

```
Shoveling through file bootcamp_example.i3.bz2                                              For help press ?
──────────────────────────────────── Frame 0: TrayInfo ────────────────────────────────────
Key:                          Type:                          Contains:                              Bytes:
2016-04-14T17:38:15.156062 I3TrayInfo                        icecube.icetray.I3TrayInfo object       3.7K

Key: 1/1        Run/Subrun: /                  Start Time:
Frame: 1/unk    Event/SubEvent: /                                        0
Stop: TrayInfo  Duration:                                                |
                                                                         IQQQQQQQQQQ
```