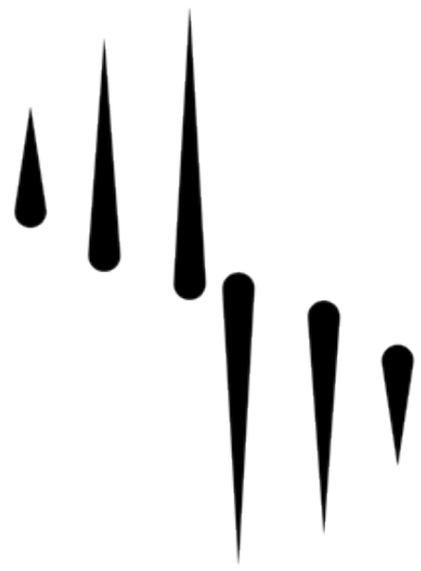


Writing an Agent From Scratch

Overview

- Start from CFilterTemplate
- Overview of the template's code
- Implementation of the sine oscillator
- Distributing your agent
- Q&A

Overview

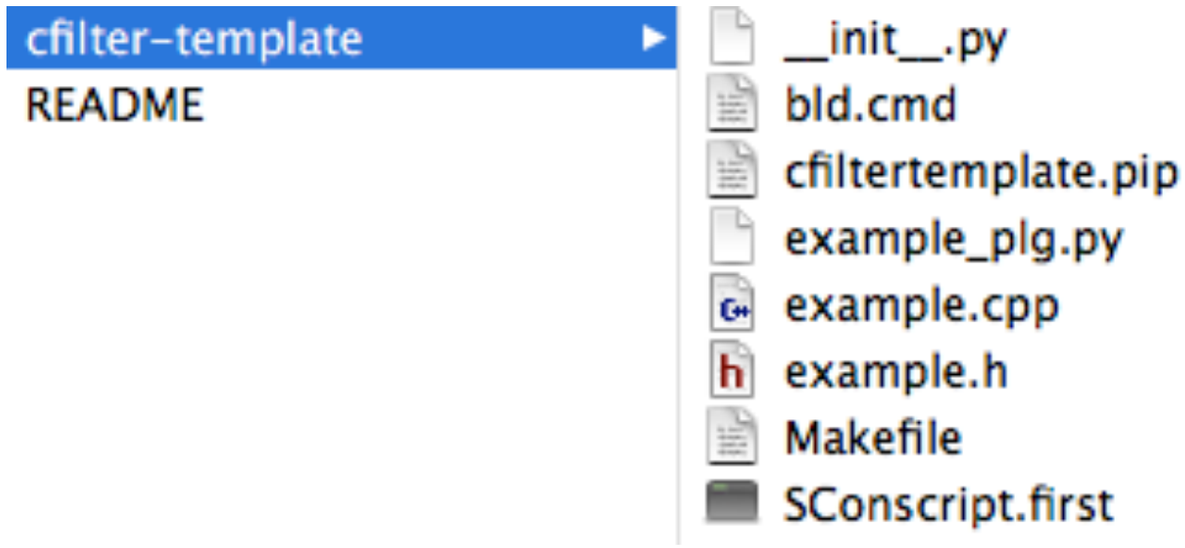
- Start from CFilterTemplate
- Overview of the template's code
- Implementation of the sine oscillator
- Distributing your agent
- Q&A

Get it from GitHub

Inside your work directory type:

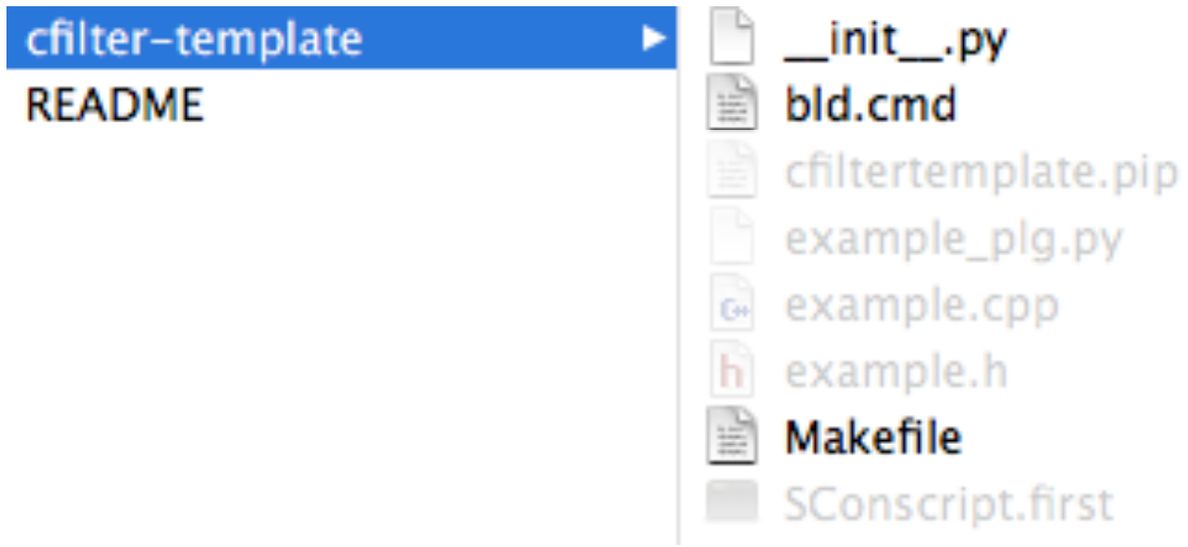
```
git clone git://github.com/Eigenlabs/EigenD-Example.git
```

You'll get the following files:



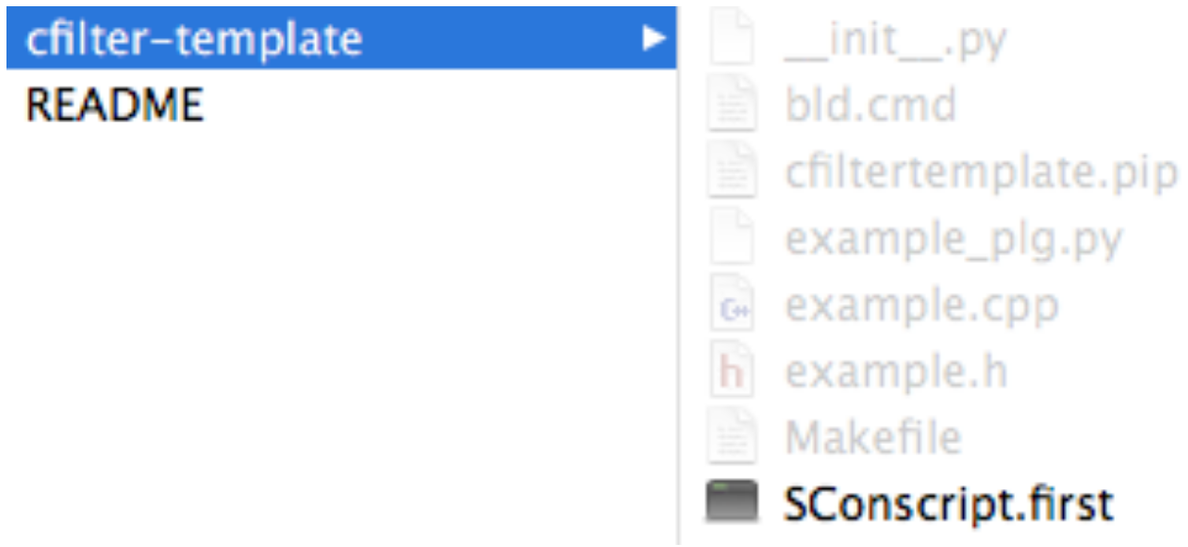
What's in the template?

- Boilerplate build system and packaging files



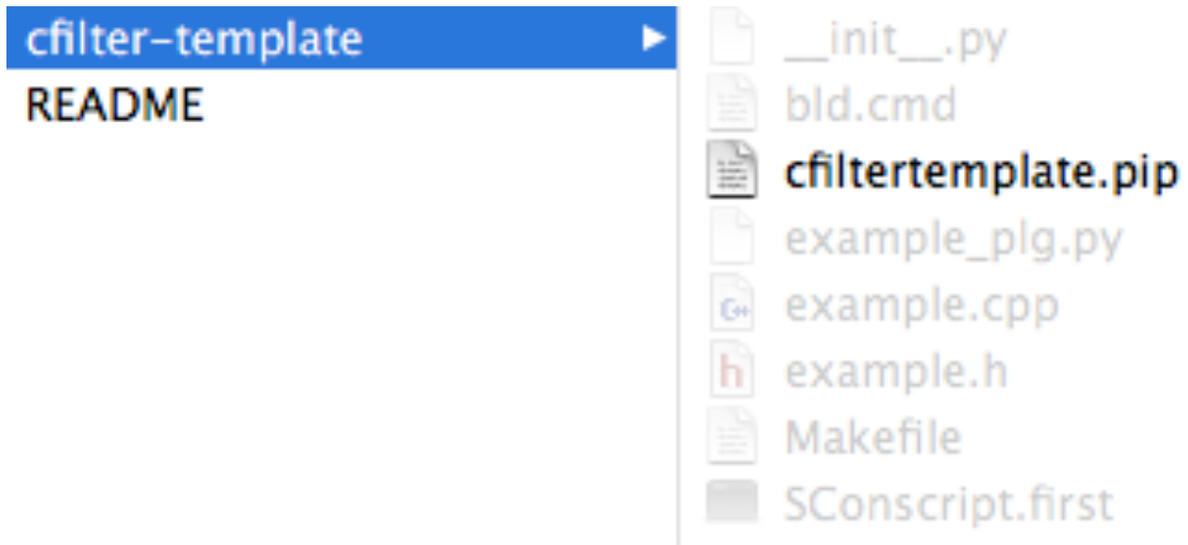
What's in the template?

- Build system configuration



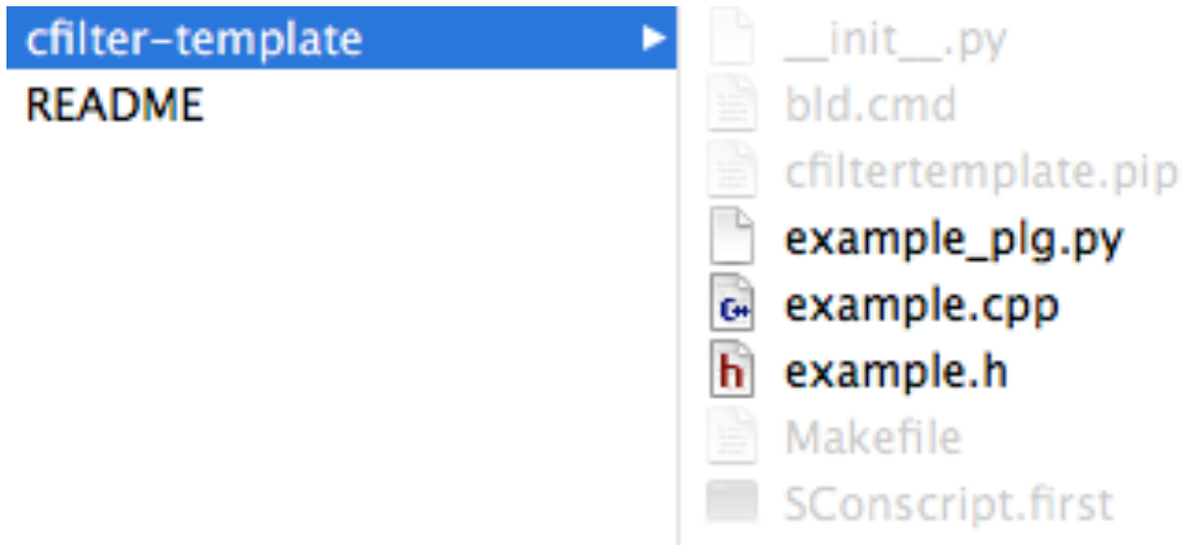
What's in the template?

- Python/C++ bindings declaration



What's in the template?

- The agent Python/C++ implementation



Let's build it!

- The build system is shipped with EigenD as of version 2.0.35-experimental
- Copy the tools directory into the project

  For Mac:

  ```
  cp -R /usr/pi/release-2.0.35-experimental/tools tools
  ```

  For Windows:

  ```
  md tools
  xcopy /e "c:\Program Files (x86)\Eigenlabs\release-2.0.35-experimental\tools" tools
  ```

- Type `make` or `bld.cmd`

See it run

- Allow EigenD to see the plugin path of your project by adding

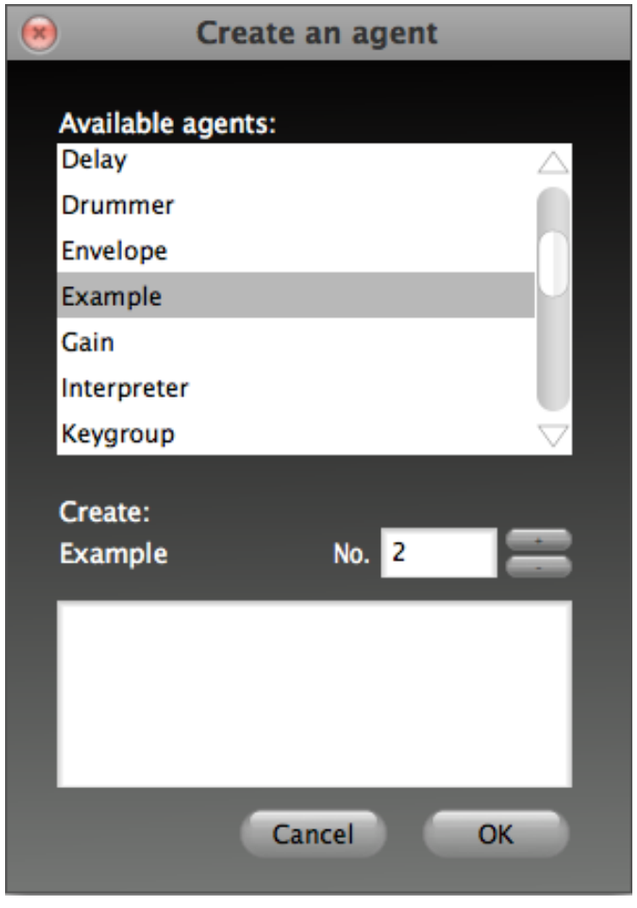
  `<path_to>/cfilter-template/tmp/plugins` to either:

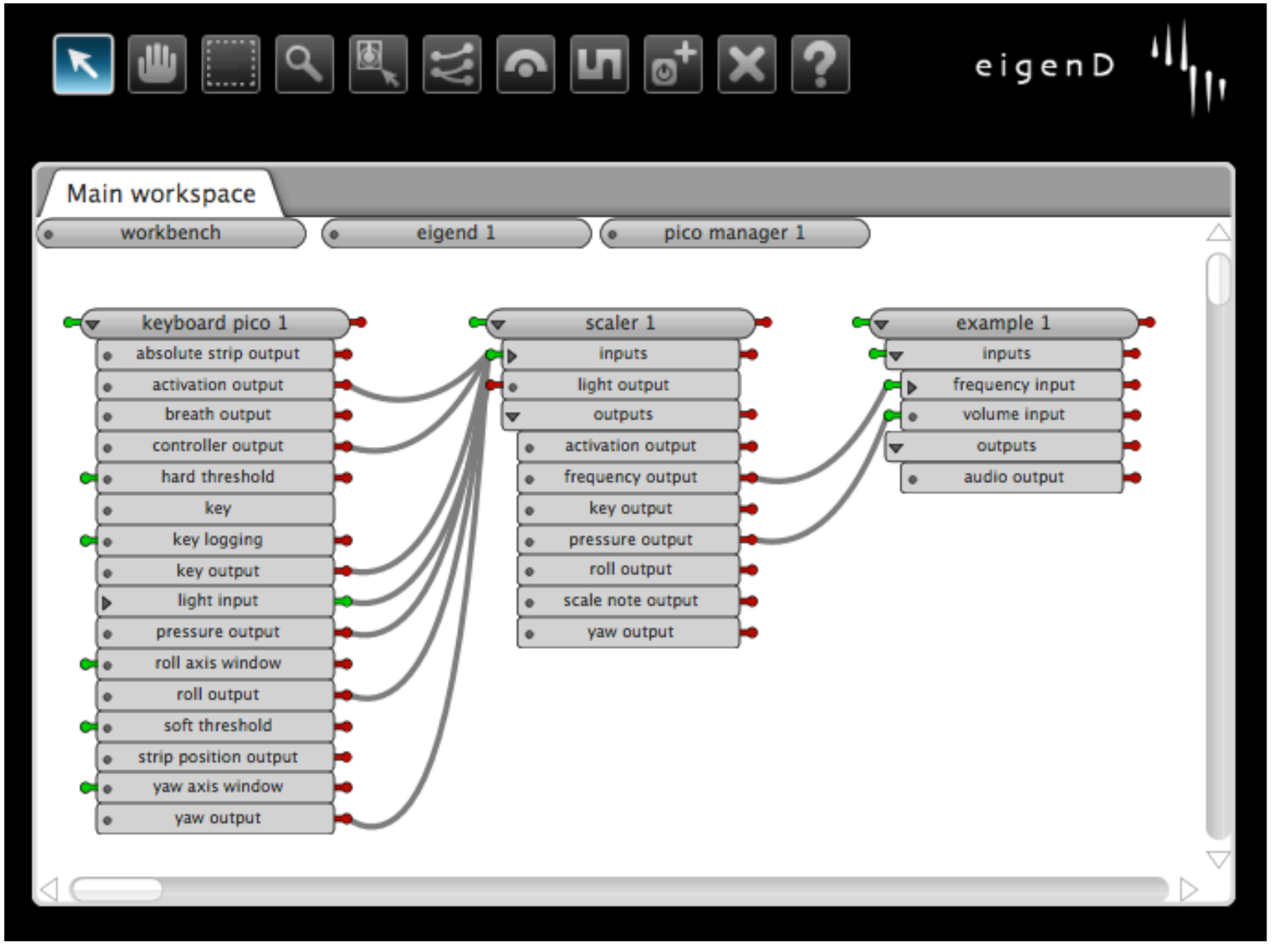
  ```
  /Users/<you>/Library/Eigenlabs/Global/paths.txt

  C:\Users\<you>\My Documents\Eigenlabs\Global
  \paths.txt
  ```

- Run EigenD and Workbench
- Create a new instance of your agent

See it run


Create an agent
Available agents:
Delay
Drummer
Envelope
Example
Gain
Interpreter
Keygroup
Create:
Example
No. 2
Cancel
OK


eigenD
Main workspace
workbench
eigend 1
pico manager 1
keyboard pico 1
absolute strip output
activation output
breath output
controller output
hard threshold
key
key logging
key output
light input
pressure output
roll axis window
roll output
soft threshold
strip position output
yaw axis window
yaw output
scaler 1
inputs
light output
outputs
activation output
frequency output
key output
pressure output
roll output
scale note output
yaw output
example 1
inputs
frequency input
volume input
outputs
audio output

Overview

- Start from CFilterTemplate
- Overview of the template's code
- Implementation of the sine oscillator
- Distributing your agent
- Q&A

The outer Python shell

- Agent metadata, ports, clocking and configuration is done through Python
- This is the simplest possible agent

```
from pi import agent
from . import example_version as version

class Agent(agent.Agent):
    def __init__(self, address, ordinal):
        agent.Agent.__init__(self, signature=version,
                             names='example', ordinal=ordinal)

        # code goes here

agent.main(Agent)
```

The outer Python shell

- Agent metadata, ports, clocking and configuration is done through Python
- This is the simplest possible agent

```
from pi import agent
from . import example_version as version
```

generated by build system

```
class Agent(agent.Agent):
    def __init__(self, address, ordinal):
        agent.Agent.__init__(self, signature=version,
                             names='example', ordinal=ordinal)

        # code goes here

agent.main(Agent)
```

The outer Python shell

- Add the clocking domain
- Create a cookie chain, which link the different elements together

```
self.domain = piw.clockdomain_ctl()

self[1] = atom.Atom(names='outputs')
self[1][1] = bundles.Output(1, True, names='audio output')
self.output = bundles.Splitter(self.domain, self[1][1])

self.example = cfiltertemplate_native.example(self.output.cookie(),
                    self.domain)

self.input = bundles.VectorInput(self.example.cookie(),
                    self.domain, signals=(1, 2))
```

The outer Python shell

- Add the clocking domain
- Create a cookie chain, which link the different elements together

```
self.domain = piw.clockdomain_ctl()

self[1] = atom.Atom(names='outputs')
self[1][1] = bundles.Output(1, True, names='audio output')
self.output = bundles.Splitter(self.domain, self[1][1])

self.example = cfiltertemplate_native.example(self.output.cookie(),
                    self.domain)

self.input = bundles.VectorInput(self.example.cookie(),
                    self.domain, signals=(1, 2))
```

The outer Python shell

- Add the clocking domain
- Create a cookie chain, which link the different elements together

```
self.domain = piw.clockdomain_ctl()

self[1] = atom.Atom(names='outputs')
self[1][1] = bundles.Output(1, True, names='audio output')
self.output = bundles.Splitter(self.domain, self[1][1])

self.example = cfiltertemplate_native.example(self.output.cookie(),
                    self.domain)

self.input = bundles.VectorInput(self.example.cookie(),
                    self.domain, signals=(1, 2))
```

The outer Python shell

- Create the input atoms for the specified signal numbers
- Specify their name, domain and policy

```
self.input = bundles.VectorInput(self.example.cookie(),
                 self.domain, signals=(1, 2))

self[2] = atom.Atom(names='inputs')
self[2][1] = atom.Atom(domain=domain.BoundedFloat(0,1),
                   names="volume input",
                   policy=self.input
                       .local_policy(1,policy.IsoStreamPolicy(1,0,0)))

self[2][2] = atom.Atom(domain=domain.BoundedFloat(0,96000, rest=440),
                   names="frequency input",
                   policy=self.input.merge_policy(2,False))
```

The outer Python shell

- Add the required Python imports

```
from pi import agent,atom,domain,policy,bundles
from . import example_version as version

import piw
import cfiltertemplate_native

class Agent(agent.Agent):
    def __init__(self, address, ordinal):
        agent.Agent.__init__(self, signature=version,
                             names='example', ordinal=ordinal)
```

The C++ declaration

```
#include <piw/piw_bundle.h>
#include <piw/piw_clock.h>

namespace cfiltertemplate
{
    class example_t
    {
        public:
            example_t(const piw::cookie_t &output,
                      piw::clockdomain_ctl_t *domain);
            ~example_t();
            piw::cookie_t cookie();

            class impl_t;
        private:
            impl_t *impl_;
    };
};
```

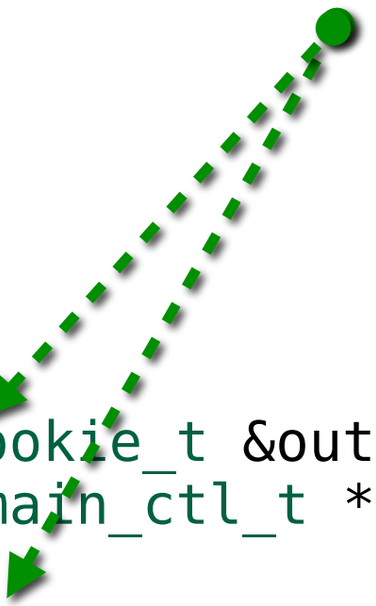
The C++ declaration

```
#include <piw/piw_bundle.h>
#include <piw/piw_clock.h>

namespace cfiltertemplate
{
    class example_t
    {
        public:
            example_t(const piw::cookie_t &output,
                      piw::clockdomain_ctl_t *domain);
            ~example_t();
            piw::cookie_t cookie();

            class impl_t;
        private:
            impl_t *impl_;
    };
};
```

the cookies that tie into the Python wrapper



The C++ declaration

```
#include <piw/piw_bundle.h>
#include <piw/piw_clock.h>

namespace cfiltertemplate
{
    class example_t
    {
        public:
            example_t(const piw::cookie_t &output,
                      piw::clockdomain_ctl_t *domain);
            ~example_t();
            piw::cookie_t cookie();

            class impl_t;
        private:
            impl_t *impl_;
    };
};
```

the implementation is delegated to a forward declaration of a private impl_t class instance that lives in the agent's definition

Python/C++ bindings

- The build system parses .pip files to create the Python to C++ wrapper automatically
- Import types and expose the relevant API

```
<<<
#include "example.h"
>>>

from piw[piw/piw.pip] import clockdomain_ctl, cookie

class example[cfiltertemplate::example_t]
{
    example(const cookie &, clockdomain_ctl *)
    cookie cookie()
}
```

Python/C++ bindings

- The build system parses .pip files to create the Python to C++ wrapper automatically
- Import types and expose the relevant API

```
<<<
#include "example.h"
>>>

from piw[piw/piw.pip] import clockdomain_ctl, cookie

class example[cfiltertemplate::example_t]
{
    example(const cookie &, clockdomain_ctl *)
    cookie cookie()
}
```

just the API that's useful in Python

What is a CFilter?

- Does all the event handling
- Processing is clocked, you'll get ticking
- Convenient data input and output
- Pass-through unprocessed signals
- Can't synthesize or drop events

C++ implementation

- Declare and define the private impl_t class

```
namespace cfiltertemplate
{
    struct example_t::impl_t: piw::cfilterctl_t, piw::cfilter_t
    {
        impl_t(const piw::cookie_t &output,
                piw::clockdomain_ctl_t *domain) :

            cfilter_t(this, output, domain) {}

        piw::cfilterfunc_t *cfilterctl_create(const piw::data_t &path)
            { return new example_func_t(); }

        unsigned long long cfilterctl_thru() { return 0; }
        unsigned long long cfilterctl_inputs() { return SIG2(1,2); }
        unsigned long long cfilterctl_outputs() { return SIG1(1); }
    };
}
```

C++ implementation

- Declare and define the private impl_t class

extends the cfilter_t class and implements cfilterctl_t to provide the required config and behavior

```
namespace cfiltertemplate
{
    struct example_t::impl_t: piw::cfilterctl_t, piw::cfilter_t
    {
        impl_t(const piw::cookie_t &output,
               piw::clockdomain_ctl_t *domain) :
            cfilter_t(this, output, domain) {}

        piw::cfilterfunc_t *cfilterctl_create(const piw::data_t &path)
            { return new example_func_t(); }

        unsigned long long cfilterctl_thru() { return 0; }
        unsigned long long cfilterctl_inputs() { return SIG2(1,2); }
        unsigned long long cfilterctl_outputs() { return SIG1(1); }
    };
}
```

C++ implementation

- Declare and define the private impl_t class

```
namespace cfiltertemplate
{
    struct example_t::impl_t: piw::cfilterctl_t, piw::cfilter_t
    {
        impl_t(const piw::cookie_t &output,
               piw::clockdomain_ctl_t *domain) :
            cfilter_t(this, output, domain) {}

        piw::cfilterfunc_t *cfilterctl_create(const piw::data_t &path)
            { return new example_func_t(); }

        unsigned long long cfilterctl_thru() { return 0; }
        unsigned long long cfilterctl_inputs() { return SIG2(1,2); }
        unsigned long long cfilterctl_outputs() { return SIG1(1); }
    };
}
```

configure the masks of signals that are passed through, used as input and sent as output

C++ implementation

- Declare and define the private impl_t class

```
namespace cfiltertemplate
{
    struct example_t::impl_t: piw::cfilterctl_t, piw::cfilter_t
    {
        impl_t(const piw::cookie_t &output,
               piw::clockdomain_ctl_t *domain) :
            cfilter_t(this, output, domain) {}

        piw::cfilterfunc_t *cfilterctl_create(const piw::data_t &path)
            { return new example_func_t(); }

        unsigned long long cfilterctl_thru() { return 0; }
        unsigned long long cfilterctl_inputs() { return SIG2(1,2); }
        unsigned long long cfilterctl_outputs() { return SIG1(1); }
    };
}
```

create a new functor every time a new wire connection is made

C++ implementation

```
struct example_func_t: piw::cfilterfunc_t
{
    example_func_t() {}

    bool cfilterfunc_start(piw::cfilterenv_t *env, const piw::data_nb_t &id)
    {
        // event start logic
        return true;
    }

    bool cfilterfunc_process(piw::cfilterenv_t *env, unsigned long long from,
            unsigned long long to, unsigned long samplerate, unsigned buffersize)
    {
        // process the signal values here
        return true;
    }

    bool cfilterfunc_end(piw::cfilterenv_t *env, unsigned long long to)
    {
        // event end logic
        return false;
    }
};
```

C++ implementation

the cfilter env instance provides various data input and output methods

```
struct example_func_t: piw::cfilterfunc_t
{
    example_func_t() {}

    bool cfilterfunc_start(piw::cfilterenv_t *env, const piw::data_nb_t &id)
    {
        // event start logic
        return true;
    }

    bool cfilterfunc_process(piw::cfilterenv_t *env, unsigned long long from,
            unsigned long long to, unsigned long samplerate, unsigned buffersize)
    {
        // process the signal values here
        return true;
    }

    bool cfilterfunc_end(piw::cfilterenv_t *env, unsigned long long to)
    {
        // event end logic
        return false;
    }
};
```

C++ implementation

```
struct example_func_t: piw::cfilterfunc_t
{
    example_func_t() {}

    bool cfilterfunc_start(piw::cfilterenv_t *env, const piw::data_nb_t &id)
    {
        // event start logic
        return true;
    }

    bool cfilterfunc_process(piw::cfilterenv_t *env, unsigned long long from,
            unsigned long long to, unsigned long samplerate, unsigned buffersize)
    {
        // process the signal values here
        return true;
    }

    bool cfilterfunc_end(piw::cfilterenv_t *env, unsigned long long to)
    {
        // event end logic
        return false;
    }
};
```

called when an event is started, the event id is passed in

C++ implementation

```
struct example_func_t: piw::cfilterfunc_t
{
    example_func_t() {}

    bool cfilterfunc_start(piw::cfilterenv_t *env, const piw::data_nb_t &id)
    {
        // event start logic
        return true;
    }

    bool cfilterfunc_process(piw::cfilterenv_t *env, unsigned long long from,
            unsigned long long to, unsigned long samplerate, unsigned buffersize)
    {
        // process the signal values here
        return true;
    }

    bool cfilterfunc_end(piw::cfilterenv_t *env, unsigned long long to)
    {
        // event end logic
        return false;
    }
};
```

true means a successful event start,
false will stop the processing of this event

C++ implementation

```
struct example_func_t: piw::cfilterfunc_t
{
    example_func_t() {}

    bool cfilterfunc_start(piw::cfilterenv_t *env, const piw::data_nb_t &id)
    {
        // event start logic
        return true;
    }
```

called every time the clock ticks
read, process and output data here

```
    bool cfilterfunc_process(piw::cfilterenv_t *env, unsigned long long from,
            unsigned long long to, unsigned long samplerate, unsigned buffersize)
    {
        // process the signal values here
        return true;
    }

    bool cfilterfunc_end(piw::cfilterenv_t *env, unsigned long long to)
    {
        // event end logic
        return false;
    }
};
```

C++ implementation

```
struct example_func_t: piw::cfilterfunc_t
{
    example_func_t() {}

    bool cfilterfunc_start(piw::cfilterenv_t *env, const piw::data_nb_t &id)
    {
        // event start logic
        return true;
    }

    bool cfilterfunc_process(piw::cfilterenv_t *env, unsigned long long from,
            unsigned long long to, unsigned long samplerate, unsigned buffersize)
    {
        // process the signal values here
        return true;
    }

    bool cfilterfunc_end(piw::cfilterenv_t *env, unsigned long long to)
    {
        // event end logic
        return false;
    }
};
```

true means the processing was successful,
false stops further processing of the event

C++ implementation

```
struct example_func_t: piw::cfilterfunc_t
{
    example_func_t() {}

    bool cfilterfunc_start(piw::cfilterenv_t *env, const piw::data_nb_t &id)
    {
        // event start logic
        return true;
    }

    bool cfilterfunc_process(piw::cfilterenv_t *env, unsigned long long from,
            unsigned long long to, unsigned long samplerate, unsigned buffersize)
    {
        // process the signal values here
        return true;
    }
```

called when the event ends

```
    bool cfilterfunc_end(piw::cfilterenv_t *env, unsigned long long to)
    {
        // event end logic
        return false;
    }
};
```

C++ implementation

```
struct example_func_t: piw::cfilterfunc_t
{
    example_func_t() {}

    bool cfilterfunc_start(piw::cfilterenv_t *env, const piw::data_nb_t &id)
    {
        // event start logic
        return true;
    }

    bool cfilterfunc_process(piw::cfilterenv_t *env, unsigned long long from,
            unsigned long long to, unsigned long samplerate, unsigned buffersize)
    {
        // process the signal values here
        return true;
    }

    bool cfilterfunc_end(piw::cfilterenv_t *env, unsigned long long to)
    {
        // event end logic
        return false;
    }
};
```

true indicates the event should linger,
false stops the event for real

Data API of cfilterenv_t

- Get next input value for one or all signals

```
bool cfilterenv_next(unsigned &signal, piw::data_nb_t &value, unsigned long long t);
bool cfilterenv_nextsig(unsigned signal, piw::data_nb_t &value, unsigned long long t);
```

- Get the more recent input value for signal

```
bool cfilterenv_latest(unsigned signal, piw::data_nb_t &value, unsigned long long t);
```

- Reset but not consume internal signal iterator

```
void cfilterenv_reset(unsigned signal, unsigned long long t);
```

- Enqueue output data for current event

```
void cfilterenv_output(unsigned output_signal, const piw::data_nb_t &data);
```

Data processing example

```
bool cfilterfunc_process(...)
{
    float *buffer_out, *scalar;
    piw::data_nb_t audio_out = piw::makenorm_nb(to, buffersize, &buffer_out, &scalar);
    memset(buffer_out, 0, buffersize*sizeof(float));
    *scalar = 0;

    unsigned signal;
    piw::data_nb_t value;
    while(env->cfilterenv_next(signal, value, to))
    {
       switch(signal)
       {
          case 1: // do stuff with signal 1
             break;
          case 2: // do stuff with signal 2
             break;
       }
    }
    // some more processing

    env->cfilterenv_output(1, audio_out); // output signal 1

    return true;
}
```

Overview

- Start from CFilterTemplate
- Overview of the template's code
- Implementation of the sine oscillator
- Distributing your agent
- Q&A

Overview

- Start from CFilterTemplate
- Overview of the template's code
- Implementation of the sine oscillator
- Distributing your agent
- Q&A

Let’s look at actual EigenD code

Overview

- Start from CFilterTemplate
- Overview of the template's code
- Implementation of the sine oscillator
- Distributing your agent
- Q&A

Use the build system

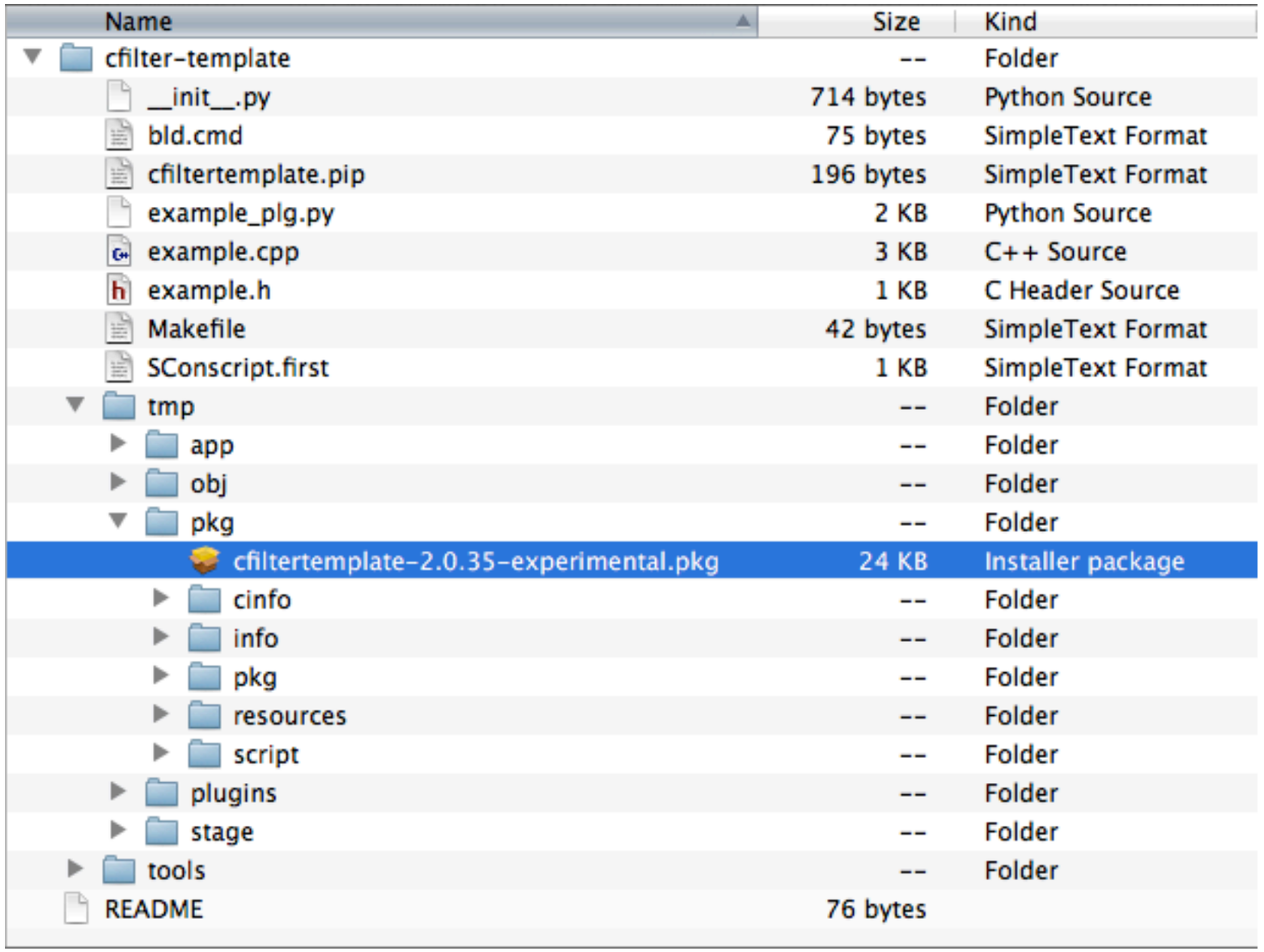
- Go to the project directory and type:

  For Mac:
  `make mpkg`

  For Windows:
  `bld.cmd target-mpkg`

- You'll get a single installer file for your agent
- Anyone can now double click, install and use your agent

The installer is in tmp/pkg



Overview

- Start from CFilterTemplate
- Overview of the template's code
- Implementation of the sine oscillator
- Distributing your agent
- Q&A