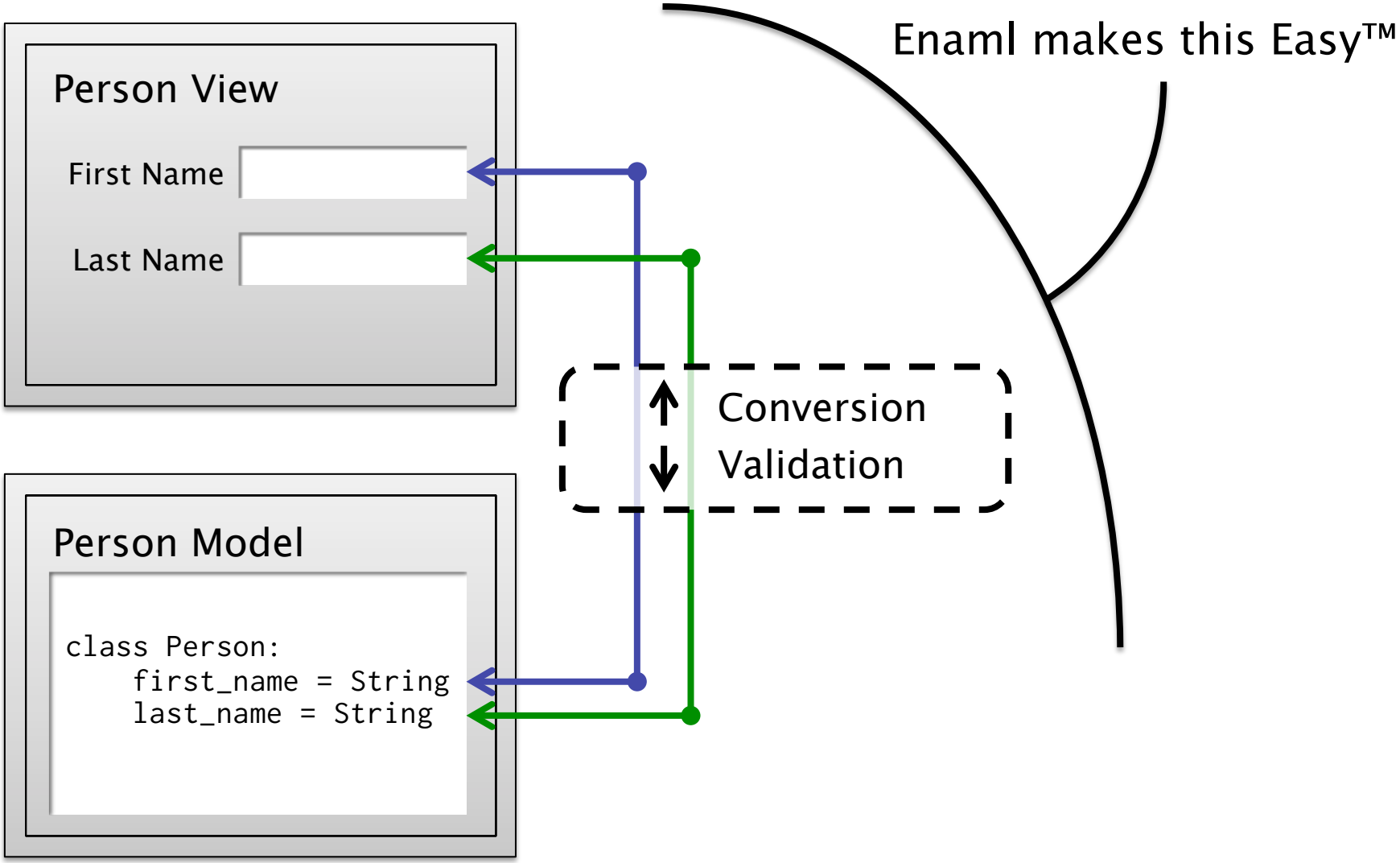

# Enaml

## GUIs Made Easy

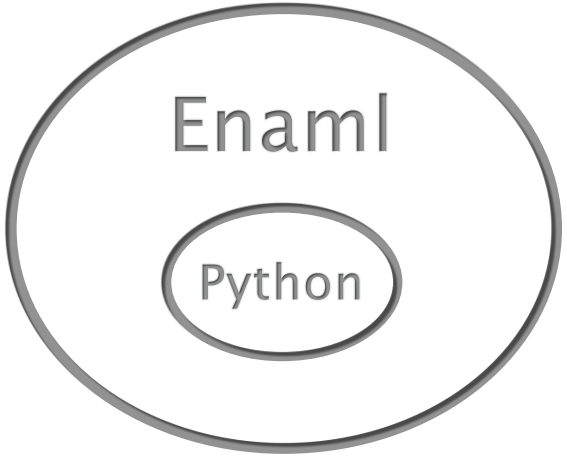
A DSL for Declarative User Interfaces

# Basic MVC Concept



# Some Existing Approaches

| Visual Studio® | Xcode® | Qt Designer® |
| --- | --- | --- |
| • XML-based markup<br>• .Net backing code<br>• Bind to properties<br>• Markup is translated | • No markup<br>• Obj-C backing code<br>• Bind to properties<br>• Code is generated | • XML markup<br>• C++ backing code<br>• Bind to signals/slots<br>• Markup is interpreted |

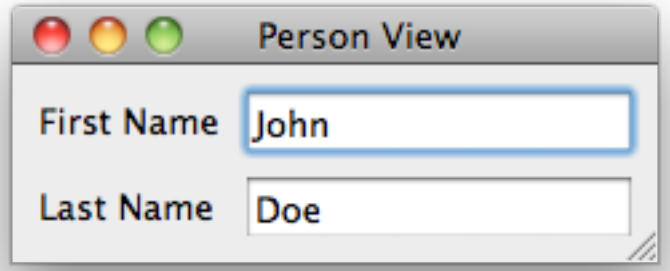



- Python-based markup language
- Strict superset of the Python language
- Bind to arbitrary Python *expressions*
- Markup is dynamically executed

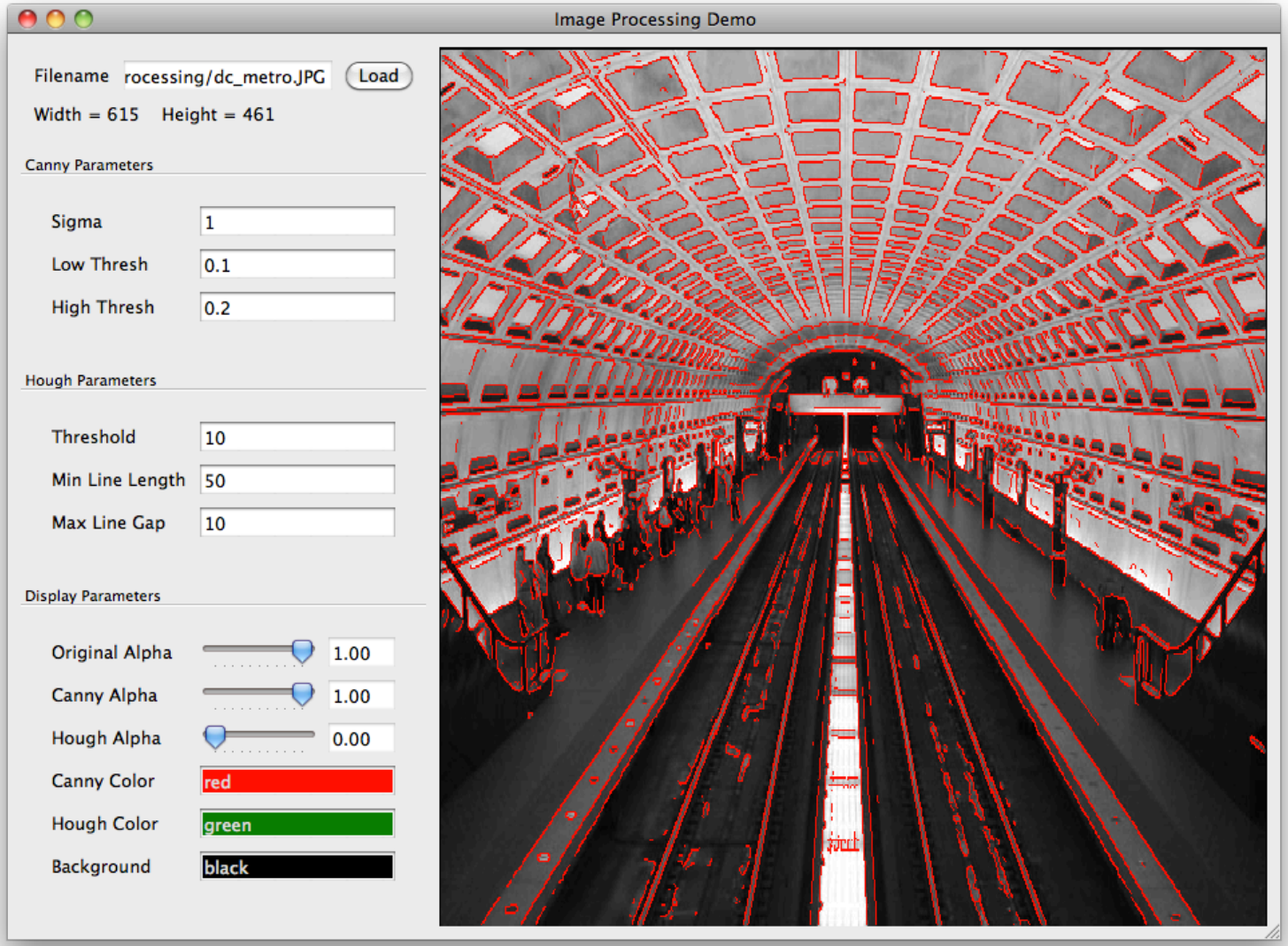
# Show Me The Code!

```python
# model.py

from traits.api import HasTraits, Str
import enaml


class Person(HasTraits):
    first_name = Str
    last_name = Str


if __name__ == '__main__':

    john = Person(first_name='John',
                  last_name='Doe')

    with enaml.imports():
        from view import View

    view = View(person=john)

    view.show()
```

```
# view.enaml

enamldef View(MainWindow):
    attr person
    title = 'Person View'
    Form:
        Label:
            text = 'First Name'
        Field:
            value := person.first_name
        Label:
            text = 'Last Name'
        Field:
            value := person.last_name
```



# Application Example ~150 LOC



# Widget Gallery ~150 LOC

Widget Gallery

Push Button | Radio One | Radio Two
Toggle Button | Check One | Check Two
thing

March 2012

| | Sun | Mon | Tue | Wed | Thu | Fri | Sat |
|---|---|---|---|---|---|---|---|
| 9 | 26 | 27 | 28 | 29 | 1 | 2 | 3 |
| 10 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 12 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 13 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 14 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

3/8/12
3/8/12 11:04 PM
42

## Simple HTML

**Cell that spans two columns:**

| Name | Telephone | |
|---|---|---|
| Tony Stark | 555 77 854 | 555 77 855 |

**Cell that spans two rows:**

| | |
|---|---|
| First Name: | Pepper Potts |
| Telephone: | 555 77 854 |
| | 555 77 855 |

One | Two | Three

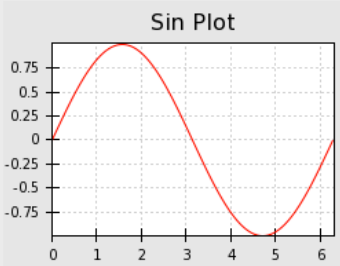
A Group Box

Very: CSS Color Name...
Simple: lightskyblue
Form Layout: indianred
Int Slider: 78
Float Slider: 0.420

| Name | Size | Last Modified |
|---|---|---|
| build | 0 kb | 02/16/12 14:49:52 |
| docs | 0 kb | 11/04/11 17:24:43 |
| enaml | 1 kb | 03/03/12 19:51:23 |
| backends | 0 kb | 03/03/12 19:32:04 |
| components | 3 kb | 03/08/12 12:49:40 |
| core | 1 kb | 03/03/12 19:51:23 |
| graphics | 0 kb | 02/16/12 14:58:55 |
| icons | 0 kb | 03/03/12 19:50:59 |
| item_models | 0 kb | 03/03/12 19:48:13 |
| layout | 0 kb | 03/03/12 19:32:04 |

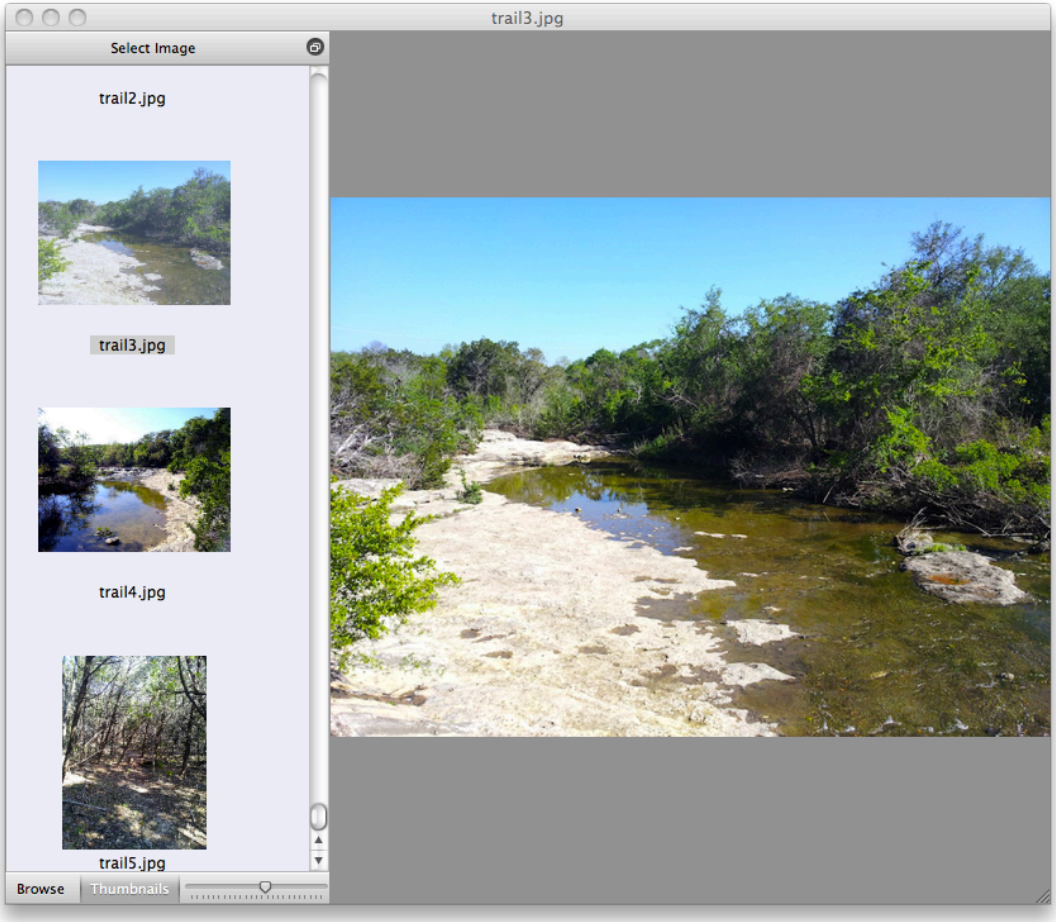
| | Name | Size | Last Modified |
|---|---|---|---|
| 1 | build | 0 kb | 02/16/12 14:4... |
| 2 | docs | 0 kb | 11/04/11 17:2... |
| 3 | enaml | 1 kb | 03/03/12 19:5... |
| 4 | enaml.egg-info | 0 kb | 11/02/11 22:4... |
| 5 | enamldoc | 0 kb | 03/03/12 19:3... |

build
docs
enaml
enaml.egg-info
enamldoc
examples
licenses
tools
foo.png
fs_model.pyc
gallery_data.py
gallery_data.pyc
lena.png
LICENSE.txt
qt_grid.py
README.rst
setup.py
state_test.py
table.py
test.enaml
test2.enaml
test_grid.enaml
tests_report.rst
traits_ex.py
widget_gallery.enaml




```
enamldef Groups(GroupBox):
    title = 'A Group Box'
    Form:
        Label:
            text = 'Very'
        Field:
            placeholder_text = 'CSS Color Name...'
            bgcolor << value
        Label:
            text = 'Simple'
        Field:
```

# Preview App ~200 LOC



# Language Structure

- Enaml is a strict superset of Python
- Any valid Python (2.x) file is a valid Enaml file
- Enaml extends the Python language with the keyword `enamldef`
- The `enamldef` keyword begins a block of Enaml code which extends Python's standard grammar and scoping rules

# Language Structure

- Enaml components are trees with dynamic bindable attributes
- The root of a component derives from another root or a builtin Enaml component and defines a new usable component type

# Language Structure

```
enamldef CustomField(Field):
    pass


enamldef ReallyCustomField(CustomField):
    pass
```

# Language Structure

- Tree branches are instances of tree roots or built-in components.
- Tree leaves are identical to tree branches but have no children.
- The distinction between branches and leaves is only conceptual, but some components do not allow children to be added to them.

# Language Structure

```
enamldef MyContainer(Container):
    CustomField:
        pass
    ReallyCustomField:
        pass
    Container:
        Field:
            pass
        PushButton:
            pass
        RadioButton:
            pass
```

# Language Structure

- Roots and branches are customized by binding to their attributes

```
enamldef Main(Window):
    title = ‘Window Title’
    Field:
        value = ‘Field Value’
```

# Language Structure

- Roots can by further customized by declaring new attributes and events

```
enamldef Main(Window):
    attr model
    event custom_event
    title = 'Window Title'
    Field:
        value = model.value
```

# Language Structure

- The grammar of declaring an attr or an event supports four different forms

```
(event|attr) <name>

(event|attr) <name>: <type>

(event|attr) <name> <binding>

(event|attr) <name>: <type> <binding>
```

# Attribute Binding

- Enaml provides five different operators which can be used to bind Python expressions to component attributes.
- These operators provide very powerful introspection and dependency tracking
- Each binding operator has its own behavioral semantics as well as restrictions on what form the RHS expression may take.

# Attribute Binding – Default

- The default operator =
- Left associative
- Single eval, no introspection
- RHS can be any expression

# Attribute Binding – Default

```
enamldef Main(Window):
    attr message = "Hello, world!"
    Container:
        Label:
            text = message
```

# Attribute Binding – Subscription

- The subscription operator <<
- Left associative
- Evals and assigns when invalid
- RHS can be any expression

# Attribute Binding – Subscription

```
import math

enamldef Main(MainWindow):
    title = 'Slider Example'
    Form:
        Label:
            text = 'Log Value'
        Field:
            value << math.log(val_slider.value)
            read_only = True
        Slider:
            id: val_slider
            tick_interval = 50
            maximum = 1000
            minimum = 1
```

# Attribute Binding – Update

- The update operator >>
- Right associative
- Pushes value on change
- RHS must be assignable expression

# Attribute Binding – Update

```
from traits.api import HasTraits, Str, on_trait_change

class Person(HasTraits):

    name = Str

    @on_trait_change('name')
    def print_name(self):
        print 'name changed', self.name

enamldef Main(Window):
    attr person = Person()
    Container:
        Field:
            value >> person.name
```

# Attribute Binding – Delegation

- The delegation operator `:=`
- Bi-Directional
- Pushes and pulls values
- RHS must be assignable expression

# Attribute Binding – Delegation

```
from traits.api import HasTraits, Str, on_trait_change

class Person(HasTraits):

    name = Str(‘John’)

    @on_trait_change('name')
    def print_name(self):
        print 'name changed', self.name

enamldef Main(Window):
    attr person = Person()
    Container:
        Field:
            value >> person.name
        Field:
            value := person.name
```

# Attribute Binding – Notification

- The notification operator `::`
- Right associative
- Executes code on change
- RHS can be any arbitrary Python code except for `def`, `class`, `return`, `yield`

# Attribute Binding – Notification

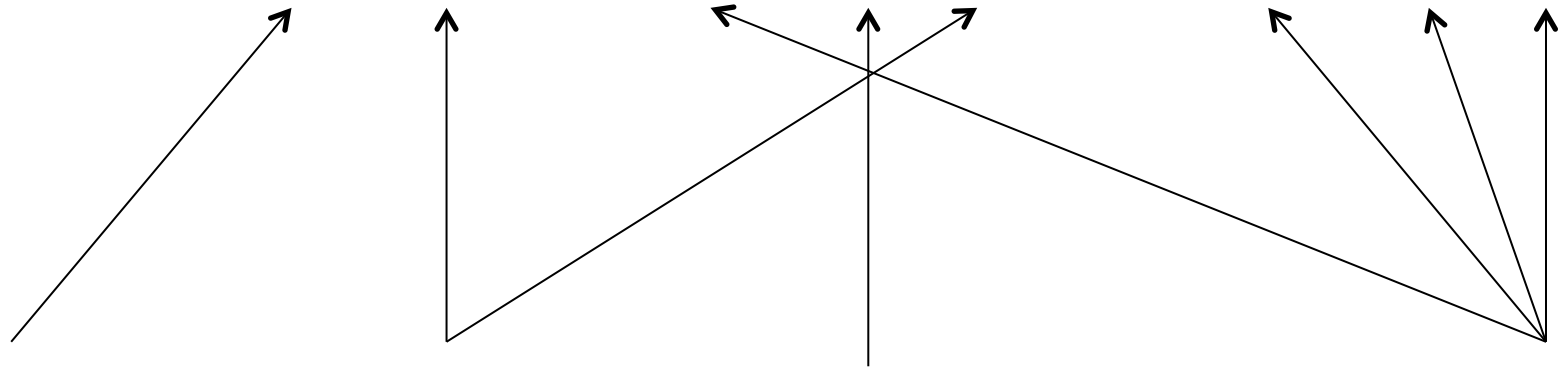
```
from traits.api import HasTraits, Str, on_trait_change

class Person(HasTraits):

    name = Str
    @on_trait_change('name')
    def print_name(self):
        print 'name changed', self.name

enamldef Main(Window):
    attr person = Person()
    Container:
        Field:
            value >> person.name
            value :: print ‘simple statement’
            value ::
                for i in range(10):
                    print ‘complex statement’, i
```

# Attribute Binding – Dependencies

- Enaml introspecting operators are extremely robust
- They can track *almost* any dependency in an expression
- This allows the user to not worry about manually hooking up notifiers; it's all automatic.

# Attribute Binding – Dependencies

```
Field:
    id: boss_info
    value << boss(school.classroom[class_id].teacher.id).info
```



Function Returns

Locals

Indexing

Attribute Access
( `getattr` works too! )

# Attribute Binding – Dependencies

- List comprehensions work too!

```
Field:
    value << ', '.join([person.name for person in people])
```

This binding will automatically track the name of every person in the list of people, as well as the contents of the list of people itself.

# Layout System

- Layout systems in GUI toolkits typically fall into two categories:
  1. They don't exist and the developer is responsible for laying out widgets
  2. They use some form of nested box model
- Given the choice, #2 is preferable, but nested box models can be painful
- We can do better

# Layout System – Constraints

- Enaml uses a constraints based layout system
- Constraints are specified as symbolic linear expressions of components
- This allows the convenience and ease of nested box models, but also the power and flexibility of manual layout

# Layout System – Constraints

- Internally, Enaml uses the Cassowary linear constraint solver to do the heavy lifting in C++
  - OSX 10.7 now uses the same library
- Enaml provides convenience factories for auto generating constraints for the most common cases
- Constraints allow us to layout the ui in ways that are not typically possible

# Questions?