On Naming

Tony Van Eerd – C++Now 2016

May 2016

CHRISTIE®

On Naming

- Be Consistent

On Naming

- Be Consistent

```
any::clear()
optional::reset()

any::empty()
optional::operator bool()
has_value() ?
```

CHRISTIE

On Naming

- Be Consistent – *Not all consistency is equal. Local before Global.*

On Naming

- Be Consistent – *Not all consistency is equal. Local before Global.*
  - Self consistency
  - Similar consistency
  - …
  - …
  - Global Consistency

CHRISTIE®

On Naming

- Be Consistent – *Not all consistency is equal. Local before Global.*
  - Self consistency
  - Similar consistency
  - …
  - …
  - Global Consistency

```
optional<float> opf = NaN;

     opf >=  opf
     opf >= *opf
    *opf >=  opf
    *opf >= *opf
```

CHRISTIE®

On Naming

- Be Consistent – *Not all consistency is equal. Local before Global.*
  - Self consistency
  - Similar consistency
  - …
  - …
  - Global Consistency

```
optional<float> opf = NaN;

    opf >=  opf   // true
    opf >= *opf   // true
   *opf >=  opf   // true
   *opf >= *opf   // false
```

On Naming

- Be Consistent – *Not all consistency is equal. Local before Global.*
  - Self consistency
  - Similar consistency
  - …
  - …
  - Global Consistency

```
optional<float> opf = NaN;

    opf >=  opf   // true
    opf >= *opf   // true
   *opf >=  opf   // true
   *opf >= *opf   // false
```

Be *Correct.*

CHRISTIE

On Naming

- Be Consistent – *Not all consistency is equal. Local before Global.*
  - Self consistency
  - Similar consistency
  - …
  - …
  - Global Consistency

```
optional<float> op;
expected<float> ex;
any             an;

       op.has_value()
       ex.has_value()
       an.has_value()
```

CHRISTIE®

On Naming

- Be Consistent – *Not all consistency is equal. Local before Global.*
  - Self consistency
  - Similar consistency
  - …
  - …
  - Global Consistency

```
optional<float> op;
expected<float> ex;
any             an;
vector<float>   vc;

          op.has_value()
          ex.has_value()
          an.has_value()
          vc.empty()
```

On Naming

- Be Consistent – *Not all consistency is equal. Local before Global.*
  - Self consistency
  - Similar consistency
  - …
  - …
  - Global Consistency

```
optional<float> op;
expected<float> ex;
any             an;

shared_ptr<float> sp;
…

       ___.reset();
```

On Naming

- Be Consistent – *Not all consistency is equal. Local before Global.*
  - Self consistency
  - Similar consistency
  - …
  - …
  - Global Consistency

```
optional<float> op;
expected<float> ex;
any             an;

vector<float> vc;
…

          >= ??  // no (one vs many)
          == ??  // yes – Regular
```

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding

CompatiblePixels

CHRISTIE®

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding

HappyPixels

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding

observer_ptr
view_ptr

“view” *“observer”*

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding

```
observer_ptr
view_ptr
```
cadged_ptr

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding

observer_ptr
view_ptr
cadged_ptr

to borrow without intent to repay

ask for or obtain (something to which one is not strictly entitled)

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?

```
observer_ptr
view_ptr
cadged_ptr
```

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?

observer_ptr
view_ptr
cadged_ptr

“view” *“observer”*

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?

“stratify”

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?

cadged_ptr

notmy_ptr ?

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives

```
cadged_ptr
notmy_ptr ?
```

CHRISTIE®

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives

```
if (!disablePopupMenu) {
    showPopupMenu();
}
```

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity

```
raw_ptr
void f(raw_ptr<Foo> p);
void g(Foo * p);
```

“f takes a raw_ptr”

“g takes a raw pointer”

CHRISTIE®

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)

```
raw_ptr
void f(raw_ptr<Foo> p);
void g(Foo * p);
```

“f takes a raw_ptr”

“g takes a raw pointer”

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)

```
std::function ?
```

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)

std::function ?

meh

CHRISTIE®

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)
- Avoid verb/noun ambiguity

```
foo.empty()
```

make_it_empty()?

is_empty()?

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid sub-concepts.

delayed_computation_range

lazy_range

CHRISTIE®

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid sub-concepts.

```
not_my_ptr
```

```
notmy_ptr
```

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality

On Naming

```
proj.getRelativePixelSize(x, y)
proj.getRelativePixelSizeInverse(x, y)
proj.getRelativeBrightness(x, y)
```

- By use or by functionality

On Naming

```
proj.getRelativePixelSize(x, y)
proj.getRelativePixelSizeInverse(x, y)
proj.getRelativeBrightness(x, y)
proj.getRelativePixelSizeInverse_orYouCouldTh
inkOfItAsRelativeBrightness_butBrightnessOnly
InfluencedByPixelSizeAndNotOtherFactors(x, y)
```

- By use or by functionality

On Naming

```
proj.getRelativePixelSize(x, y)
proj.getRelativePixelSizeInverse(x, y)

proj.getRelativeBrightness(x,y)
```

What proj knows

```
getRelativeBrightness(proj, x, y)
```

What I know locally

- By use or by functionality

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality

On Naming

- Be Consistent
- NOT understanding is better
- Co-opt a term?
- Avoid negatives – thus avoi
- Avoid spoken ambiguity (or
- Avoid verb/noun ambiguity
- Be Concise – conceptually.
- By use or by functionality
- Be *Glaringly* Inconsistent

```
optional<float> op;
expected<float> ex;
any             an;

       op.has_value()
       ex.has_value()
       an.has_value()
```

On Naming

- Be Consistent
- NOT understanding is better
- Co-opt a term?
- Avoid negatives – thus avoid
- Avoid spoken ambiguity (or
- Avoid verb/noun ambiguity
- Be Concise – conceptually. A
- By use or by functionality
- Be *Glaringly* Inconsistent

```
optional<float> op;
expected<float> ex;
any             an;
variant<float,int>  vr;

      op.has_value()
      ex.has_value()
      an.has_value()
```

On Naming

- Be Consistent
- NOT understanding is better
- Co-opt a term?
- Avoid negatives – thus avoic
- Avoid spoken ambiguity (or
- Avoid verb/noun ambiguity
- Be Concise – conceptually.
- By use or by functionality
- Be *Glaringly* Inconsistent

```
optional<float> op;
expected<float> ex;
any             an;
variant<float,int>  vr;

      op.has_value()
      ex.has_value()
      an.has_value()
 vr.valueless_by_exception()
```

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality
- Be *Glaringly* Inconsistent

`has_value()` vs `empty()`

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality
- Be *Glaringly* Inconsistent

```
atomic<int> +=
```

CHRISTIE®

On Naming

From: Bjarne Stroustrup <bjarne@stroustrup.com>
Subject: [c++std-lib-ext-2433] Re: "corrupted" in function name

On 10/31/2015 1:35 PM, Tony Van Eerd wrote:
> Consistency may be the number one rule of API design. (For example, I am currently helping with a paper to
> make variant, any, and optional more consistent. any.clear()? optional.reset()?)
>
> Once you have some consistency, INconsistency can be a powerful tool used for good. When you make
> things inconsistent, make them jarringly, glaringly inconsistent[*] and only for good reason. Then when the
> programmer sees the inconsistency, they will hopefully know that there is a good reason behind it, and seek
> to learn that reason.

Nice rule of thumb

- Be *Glaringly* Inconsistent

On Naming

- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality
- Be *Glaringly* Inconsistent

On Naming

- Describe the thing in detail – what words did you use?
- Be Consistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality
- Be *Glaringly* Inconsistent

CHRISTIE®