

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

A Proper Property

Gašper Ažman

November 27, 2017

About Me

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

- Gašper Ažman (twitter: @atomgalaxy)
- Started teaching C++ in highschool
- Did computer vision research at Berkeley
- Helped with Amazon Search Infrastructure at a9.com
- Currently at Citadel, building really cool research tools.
- A regular at the British Standards Institute (BSI) Meetings
- On the programming committees of CppCon and C++Now.

Disclaimer

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

The opinions in this talk are my own, and do not necessarily reflect the opinions of Citadel LLC or any of its subsidiaries. In addition, no Citadel resources were used in the development of this talk.

So, What is a Property?

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

A property pretends to be a field.

Assignment:

```
1 airplane.hold = "hot air";
```

Reading:

```
1 Payload x = airplane.hold;
```

(Aside: we need a payload, and strings do nicely.)

```
1 // books truly are the greatest gift  
2 using Payload = std::string;
```

Totally.

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

If it quacks like a field, it has to be...

```
1  class Airplane {  
2      Payload hold;  
3  } airplane;
```

... a field, right?

Have you heard of this?

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN



It is only a shell...

Instead, it's a pair of a Getter and a Setter on a member.

```
1 class Airplane {  
2     struct Cargo {  
3         // Setter - assignment from Payload  
4         void operator=(Payload crate) {  
5             return {};  
6         }  
7         // Getter - implicit conversion to Payload  
8         operator Payload() const {  
9             return {};  
10        }  
11    };  
12    Cargo hold;  
13 } airplane;
```

(We need at least one byte to give the hold an address).

Let's Get More Interesting

A Proper
Property

Gašper
Ažman

```
1  class Airplane {  
2      bool hatch_closed ;  
3  public:
```

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

Let's Get More Interesting

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

```
1  class Airplane {
2      bool hatch_closed;
3  public:
4      struct Cargo {
5          Payload payload;
6          void operator=(Payload crate) {
7              if (hatch_closed) throw ClosedError{};
8              payload = std::move(crate);
9          }
8 }
```

Let's Get More Interesting

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

```
1  class Airplane {
2      bool hatch_closed;
3  public:
4      struct Cargo {
5          Payload payload;
6          void operator=(Payload crate) {
7              if (hatch_closed) throw ClosedError{};
8              payload = std::move(crate);
9          }
10         operator Payload const&() const {
11             if (hatch_closed) throw ClosedError{};
12             return payload;
13         }
14     }
```

Let's Get More Interesting

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

```
1  class Airplane {
2      bool hatch_closed;
3  public:
4      struct Cargo {
5          Payload payload;
6          void operator=(Payload crate) {
7              if (hatch_closed) throw ClosedError{};
8              payload = std::move(crate);
9          }
10         operator Payload const&() const {
11             if (hatch_closed) throw ClosedError{};
12             return payload;
13         }
14     } hold;
15 } airplane;
```

Let's Get More Interesting

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

```
1  class Airplane {
2      bool hatch_closed;
3  public:
4      struct Cargo {
5          Payload payload;
6          void operator=(Payload crate) {
7              if (hatch_closed) throw ClosedError{};
8              payload = std::move(crate);
9          }
10         operator Payload const&() const {
11             if (hatch_closed) throw ClosedError{};
12             return payload;
13         }
14     } hold;
15 } airplane;
```

Pro: this solution is pretty.

Let's Get More Interesting

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

```
1  class Airplane {
2      bool hatch_closed;
3  public:
4      struct Cargo {
5          Payload payload;
6          void operator=(Payload crate) {
7              if (hatch_closed) throw ClosedError{};
8              payload = std::move(crate);
9          }
10         operator Payload const&() const {
11             if (hatch_closed) throw ClosedError{};
12             return payload;
13         }
14     } hold;
15 } airplane;
```

Pro: this solution is pretty.

Con: it is not a solution. (It does not compile.)

But WHY?

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

We need (Airplane*) this

Not (Cargo*) this.

But... We wants it! We needs it, precious!

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

No.

You're not getting a second breakfast... I mean, a second **this**.

The Problem

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

Get The Host's this.
... while being reasonably easy to use.

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

Attempt 1:

Keep It Surprisingly Simple

Store the this pointer

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

```
1  class Airplane {
2      bool hatch_closed;
3  public:
4      struct Cargo {
5          Payload payload;
6          void operator=(Payload crate) {
7              if (host->hatch_closed) throw ClosedError{};
8              payload = std::move(crate);
9          }
10         operator Payload const&() const {
11             if (host->hatch_closed) throw ClosedError{};
12             return payload;
13         }
14         Airplane* host;
15     } hold;
16     Airplane() : hold{this} {} // every. time.
17 } airplane;
```

So... How'd we do?

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

Poorly.

Needs extra space? Check.

Error-prone? Check.

No help from C++? Check.

Easy? To understand, yes. To maintain? Good luck.
(does not pass the WWTD¹ test)

¹What Would The Default Constructor Do 

Moving the Goalposts Much?

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

**Solution
Criteria**

A Better
Idea

Preview

SIITT

Synthesis

FIN

We need some criteria.

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

**Solution
Criteria**

A Better
Idea

Preview

SIITT

Synthesis

FIN

The First Rule of C++

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

**Solution
Criteria**

A Better
Idea

Preview

SIITT

Synthesis

FIN

The First Rule of C++
We only pay for what we use.

Criteria

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

**Solution
Criteria**

A Better
Idea

Preview

SIITT

Synthesis

FIN

The First Rule of C++

We only pay for what we use.

At Most One Macro Per Property

The generated code must be contiguous.

The First Rule of C++

We only pay for what we use.

At Most One Macro Per Property

The generated code must be contiguous.

No Pitfalls

- Easy to read
- Easy to write
- Easy to modify

The First Rule of C++

We only pay for what we use.

At Most One Macro Per Property

The generated code must be contiguous.

No Pitfalls

- Easy to read
- Easy to write
- Easy to modify

Boils down to *Don't repeat yourself*.

And we had to repeat ourselves with every constructor and assignment operator.

A Proper Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

**A Better
Idea**

Preview

SIITT

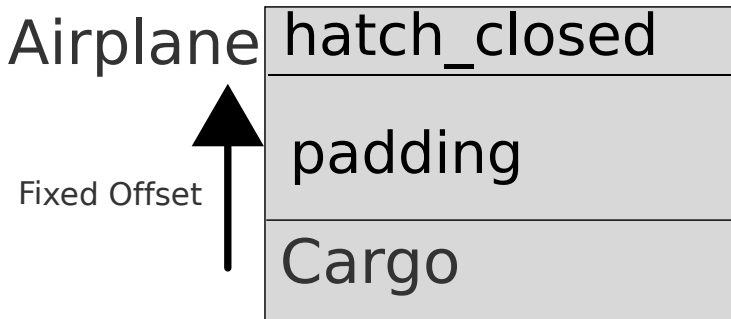
Synthesis

FIN

Attempt 2:
offsetof

Member Offsets are Constant

We already have (Cargo*) this. We can **compute** (Airplane*) this.



```
x86_64, clang: &Airplane::hold == 8
```

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

Easy Peasy!

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

We have:

```
1  //
2      auto offset = offsetof(Airplane, hold);
3      auto fixed  =
4                               this - offset;
5      auto host   =                               fixed ;
6  //
```

Now For Something That Actually Works

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

With the casts in:

```
1 //
2     auto offset = offsetof(Airplane, hold);
3     auto fixed  =
4         reinterpret_cast<char*>(this) - offset;
5     auto host    = reinterpret_cast<Airplane*>(fixed);
6 //
```

If You Like It, Put It In A Function

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

With a function around it:

```
1  Airplane* get_host() {  
2      auto offset = offsetof(Airplane, hold);  
3      auto fixed  =  
4          reinterpret_cast<char*>(this) - offset;  
5      return      reinterpret_cast<Airplane*>(fixed);  
6  }
```

But...

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

But, it has warnings!

```
offset_of.cpp:34:23: warning: offset of  
    on non-standard-layout type 'Airplane'  
    [-Winvalid-offsetof]
```

```
auto offset    = offsetof(Airplane, hold);  
                ^           ~~~~
```

(No, it's not UB, if you're using c++17)

But, Is it... Legal?

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

Turns out this doesn't have a warning, and is better than
`constexpr`.

```
1  std::integral_constant<size_t,  
2      offsetof(Airplane, cargo)>::value;
```


Why is this OK?

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

Because there can never be anything virtual between a member and its *directly enclosing class*.

Think of a non-virtual member function. It uses the `this` pointer and an offset to get to the member, because there is no vtable. That offset has to be constant.

What we are doing has no chance of not working.

So... How'd we do?

A Proper Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

We did OK. For a once-off.

- The getter and setter pair are completely ad-hoc.
- Ad-Hoc `get_host()` function with scary casts.

How About This?

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

```
1  class Airplane {
2      struct Cargo {
3          Payload payload;
4          template <typename Host>
5          void set(Host& host, Payload crate) {
6              if (host.hatch_closed) raise ClosedError{};
7              payload = std::move(crate);
8          }
9          template <typename Host>
10         auto const& get(Host const& host) const {
11             if (host.hatch_closed) raise ClosedError{};
12             return payload;
13         }
14     };
15 public:
16     bool hatch_closed;
17     LIBPROPERTY_WRAP((Cargo), hold, Airplane);
18 };
```

The Anticlimax

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

I don't think we can get away without macros.

The Anticlimax

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

I don't think we can get away without macros.
But, I promise they're not the worst thing about this solution.

The Anticlimax

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

I don't think we can get away without macros.
But, I promise they're not the worst thing about this solution.
Wait, that's not a good thing.

The Anticlimax

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

I don't think we can get away without macros.
But, I promise they're not the worst thing about this solution.

Wait, that's not a good thing.

... well, maybe they are.

Down The Rabbit-Hole

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

With `get_host()`, we had:

```
1  class Airplane {
2      struct Cargo {
3          Payload payload;
4          void operator=(Payload crate) {
5              auto& host = *Airplane::get_host(this);
6              /* use host.hatch_closed */
7          }
8      };
9  public:
10     bool hatch_closed;
11     Cargo hold;
12
13     static Airplane* get_host(Cargo* cargo) {
14         return /* cargo - offsetof(Airplane, cargo); */
15     }
16 };
```


To Make A Macro

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

We need contiguous, repetitive, easily substitutable code.

Also, *sure, we had one cargo, but...*

But What About Second Cargo?

This is pretty doable:

```
1 class Airplane {
2     template <auto closed>
3     struct Cargo {
4         Payload payload;
5         void operator=(Payload crate) {
6             auto const& host = Airplane::get_host(*this);
7             if (host.*closed) throw ClosedError{};
8             payload = std::move(crate);
9         }
10    };
11 public:
12    bool hold_closed;
13    bool frunk_closed;
14    Cargo<&Airplane::hold_closed> hold;
15    static Airplane& get_host(decltype(hold)&);
16    Cargo<&Airplane::frunk_closed> frunk;
17    static Airplane& get_host(decltype(frunk)&);
```

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

OK Now?

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

Better, but we still need to:

- manually get the host pointer.
- We need at least 3 overloads of `get_host`: (`const&`, `&`, `&&`) per member.
- We need to manually type `Airplane::`
- `get_host()` pollutes the interface of `Airplane`, and choosing an uglier and less-likely-to-clash name makes our implementation uglier too.
- What if we only had one hatch?

Store It In The Type

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

Attempt 3:

Store It In The Type.

But What About Second Cargo, Reprise?

A Proper
Property

Gašper
Ažman

What if there is only one hatch?

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

```
1  class Airplane {
2      template <auto closed>
3      struct Cargo { Payload payload; /* */ };
4  public:
5      bool hatch_closed;
6      Cargo<&Airplane::hatch_closed> hold;
7      Cargo<&Airplane::hatch_closed> frunk;
8      static Airplane& get_host(decltype(hold) &);
9      // ERROR: same type
10     static Airplane& get_host(decltype(frunk) &);
11 }
```

But What About Second Cargo, Reprise II?

A Proper
Property

Gašper
Ažman

Type tags to the rescue!

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIIT

Synthesis

FIN

```
1  class Airplane {
2      template <auto closed, typename Tag >
3      struct Cargo { Payload payload; /* */ };
4  public:
5      bool hatch_closed;
6      struct hold_tag;
7      Cargo<&Airplane::hatch_closed, hold_tag> hold;
8      struct frunk_tag;
9      Cargo<&Airplane::hatch_closed, frunk_tag> frunk;
10     static Airplane& get_host(decltype(hold)&);
11     // Now fine.
12     static Airplane& get_host(decltype(frunk)&);
13 };
```

Type Tags Are Awesome?

But wait, there's more!

```
1 class Airplane {
2     template <auto closed, typename Tag>
3     struct Cargo { Payload payload; /* */ };
4 public:
5     bool hatch_closed;
6     struct hold_tag {
7         static auto offset_of() {
8             // Airplane is fully defined, unless it isn't
9             // when *called*
10            return std::integral_constant<size_t,
11                offsetof(Airplane, hold)>{};
12        }
13    };
14    Cargo<&Airplane::hatch_closed, hold_tag> hold;
15    struct frunk_tag { /* same as above */ };
16    Cargo<&Airplane::hatch_closed, frunk_tag> frunk;
17};
```

In Other News, Stack Corruption.

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

Also, what about this?

```
1  Airplane airplane;  
2  auto x = airplane.cargo; // works!  
3  x = "foo";               // corrupts the stack
```


Fix

A Proper Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

```
1  template <auto managed>
2  struct Cargo {
3      friend Airplane ;
4      /* getters, setters */
5      private:
6      Cargo() = default;
7      Cargo(Cargo const&) = default;
8      Cargo(Cargo&&) = default;
9      Cargo const& operator=(Cargo const&) = default;
10     Cargo&& operator=(Cargo&&) = default;
11     ~Cargo() = default;
12 };
```

Now only Airplane can manage Cargo.

... Better.

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

Now Breaks:

```
1 Airplane airplane;  
2 // breaks, copy constructor is private.  
3 auto x = airplane.cargo;
```

Really Though?

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

This is a lot of code. We want to put this into a library.

It gets to be *a lot more code* when you want return-type deduction, SFINAE, and templates for getters and setters to work correctly.

So, What do We Need?

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

We need to wrap our getter/setter provider into an adaptor.

```
1  template <typename Property, typename Tag>
2  class wrapper {
3      // allow 'host' to access self::value
4      friend host;
5      Property value;
6
7      constexpr wrapper()=default;
8      constexpr wrapper(wrapper const&)=default;
9      constexpr wrapper(wrapper&&)=default;
10     ~wrapper()=default;
11     constexpr wrapper& operator=(wrapper const&)=
        default;
12     constexpr wrapper& operator=(wrapper&&)=default;
```

Setters

A Proper Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

```
1  template <typename Property, typename Tag>
2  class wrapper { /*cont*/
3      /* setter implementation */
4      template <typename X>
5      decltype(auto) operator=(X&& val) & {
6          return value.set(
7              ::libproperty::impl::get_host(*this),
8              std::forward<X>(val));
9      }
10     /* and the const& and && variants */
```

Getters

A Proper Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

```
1  template <typename Property, typename Tag>
2  class wrapper { /*cont*/
3      // SFINAE-detect getter presence
4      // also: you must say it 3 times (Vittorio,
        thanks!)
5      template <typename V = value_type, // fake params
6                  typename H = host,
7                  typename = decltype(
8                      std::declval<V>().get(
9                          std::declval<H const&>()))>
10     auto get() const & -> decltype(auto)
11     {
12         return value.get(
13             ::libproperty::impl::get_host(*this));
14     }
15     /* and the & and && variants */
```

Implicit Conversions

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

```
1  template <typename Property, typename Tag>
2  class wrapper { /*cont*/
3      // SFINAE-detect get() presence...
4      // also: you must say it 3 times (Vittorio,
5          thanks!)
6      /* and the & and && variants */
7      template <typename W = wrapper> // type-dependent
8      operator decltype(
9          std::declval<W const&>().get())() const &
10     {
11         return get();
12     }
```

The Magic Macro

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

```
1  #define LIBPROPERTY_WRAP(type, name, host) \
2      LIBPROPERTY__DECLARE_TAG(name, host); \
3      ::libproperty::wrapper< \
4          LIBPROPERTY__PARENTHESTIZED_TYPE type, \
5          host::LIBPROPERTY__TAG_NAME(name)> \
6          name; \
7      static_assert("require semicolon")
8
9  #define LIBPROPERTY__DECLARE_TAG(name, host) \
10     struct _libproperty__##name##_prop_tag { \
11         using host_type = host; \
12         auto static constexpr offset() \
13         { \
14             return std::integral_constant<size_t, \
15                 offsetof(host, name)>{}; \
16         } \
17     }; \
18     static_assert("require semicolon")
```


That's it!

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

There are a few helpers to ferry data to-and-fro.

The trick is really in defining the tag type `*outside*` the property, so we can reuse our wrapper.

The *other* trick is doing the `offsetof` inside an auto-typed `constexpr` function that returns an integral constant.

This defers lookup until all the types of all data members are known.

So, This Works!

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

We have avoided many, many hours of "you can't do this because the type isn't defined yet" with this path.

Libproperty has a lot more cool features:

- You can get the host pointer as a universal reference, and thus only have to write one getter template.
- The host is specified once, in the macro.
- You can store the value **in** the Cargo object. We could do that here too, and avoided the space penalty, but there are pitfalls.
- If you want comparisons with strings to work, you need to overload all of them - the library forwards those for you.

Questions?

A Proper
Property

Gašper
Ažman

Author

Disclaimer

Recap

The Problem

KISS

Solution
Criteria

A Better
Idea

Preview

SIITT

Synthesis

FIN

Thank You.