xproperty

Johan Mabille and Sylvain Corlay

Jan 21, 2021

INSTALLATION

C++ properties and observer pattern

`xproperty` is a C++ library providing traitlets-style properties.

`xproperty` provides an implementation of the observer patterns relying on C++ template and preprocessor metaprogramming techniques.

Properies of observed objects have no additional memory footprint than the value they hold. The assignment of a new value is simply replaced at compiled time by

- the call to the validator for that property
- the actual underlying assigment
- the call to the observor for that property.

We also provide the implementation of an `xobserved` class whose static validator and observer are bound to a dynamic unordered map of callbacks that can be registered dynamically.

`xproperty` requires a modern C++ compiler supporting C++14. The following C++ compilers are supported:

- On Windows platforms, Visual C++ 2015 Update 2, or more recent
- On Unix platforms, gcc 4.9 or a recent version of Clang

CHAPTER

ONE

LICENSING

We use a shared copyright model that enables all contributors to maintain the copyright on their contributions.

This software is licensed under the BSD-3-Clause license. See the LICENSE file for details.

1.1 Installation

xproperty is a header-only library. We provide a package for the conda package manager.

```
conda install -c conda-forge xproperty
```

Or you can directly install it from the sources:

```
cmake -D CMAKE_INSTALL_PREFIX=your_install_prefix
make install
```

1.2 Usage

1.2.1 Basic Usage

- Declaring an observed object `Foo` with two properties named `bar` and `baz` of type *double*.
- Registering a validator, executed prior to assignment, which can potentially coerce the proposed value.
- Registering a notifier, executed after the assignement.

```
#include <iostream>
#include <stdexcept>
#include <string>

#include "xproperty/xobserved.hpp"

struct Foo : public xp::xobserved<Foo>
{
    XPROPERTY(double, Foo, bar);
    XPROPERTY(std::string, Foo, baz);
};
```

Registering an observer and a validator

```
Foo foo;

XOBSERVE(foo, bar, [](const Foo& f) {
    std::cout << "Observer: New value of bar: " << f.bar << std::endl;
});

XVALIDATE(foo, bar, [](Foo&, double proposal) {
    std::cout << "Validator: Proposal: " << proposal << std::endl;
    if (proposal < 0)
    {
        throw std::runtime_error("Only non-negative values are valid.");
    }
    return proposal;
});
```

Testing the validated and observed properties

```
foo.bar = 1.0;                              // Assigning a valid value
                                            // The notifier prints "Observer: New value␣
↪of bar: 1"
std::cout << foo.bar << std::endl;          // Outputs 1.0

try
{
    foo.bar = -1.0;                         // Assigning an invalid value
}
catch (...)
{
    std::cout << foo.bar << std::endl;      // Still outputs 1.0
}
```

Shortcuts to link properties of observed objects

```
// Create two observed objects
Foo source, target;
source.bar = 1.0;

// Link `source.bar` and `target.bar`
XDLINK(source, bar, target, bar);

source.bar = 2.0;
std::cout << target.bar << std::endl;     // Outputs 2.0
```

Out-of-order initialization of properties

```
auto foo = Foo()
    .baz("hello, world");

std::cout << foo.baz << std::endl;        // Outputs hello, world
```