boost
C++ LIBRARIES

Boost.Signals2

Dmitri Nesteruk
http://activemesa.com
dn@activemesa.com

pluralsight
hardcore developer training

Overview

- **Observer pattern a.k.a. Events**
  - Component A wants to be notified when component B does something
  - Typical example:
    knowing when a value has changed and updating the UI
- **Events are a built-in construct in some languages (e.g., C#)**
- **Publish & subscribe mechanism**
  - A class can publish a particular event, e.g. 'NameChanged'
  - Other classes can choose to receive notifications of when a name is changed
  - When the name is actually changed, *all* subscribers get notified
    (multicast)
- **Boost.Signals2 supports this mechanism**
  - Signals and Slots

Fundamentals

- **`boost::signal<T>`**
  - A signal that can be sent to anyone willing to listen
  - T is the type of the slot function
- **A slot is the function that receives the signal**
  - Ordinary function
  - Functor, `std::function`
  - Lambda
- **Connection**
  - `signal<void()> s;`
    creates a signal
  - ```
    auto c = s.connect([](){
    cout << "test" << endl;
    });
    ```
    connects the signal to the slot
- **More than one slot can be connected to a signal**
- **Disconnection**
  - `c.disconnect();`
  - Disconnects *all* slots
- **Slots can be blocked**
  - Temporarily disabled but not permanently disconnected
  - Used to prevent infinite recursion
  - `shared_connection_block(c);`
  - Unblocked when block is destroyed (e.g., out of scope) or explicitly via `block.unblock();`
    ```

Customization

- **Slots can have priority**
  - A single integer or `at_front/at_back`
  - Passed as parameter to `connect()`
- **Scoped connection**
  - `scoped_connection(c)`
  - Connects signal and slot until it goes out of scope
- **Disconnect specific slot**
  - `c.disconnect(&foo);`
  - Object must have accessible == operator
- **Lifetime tracking**
  - Keep the connection alive only while the source is alive
  - Explicitly create `slot_type` and use `track()`
- **`slot_type` is derived from the signal template argument**
  - `signal<T>::slot_type`
  - Can be passed as parameter
  - Signal owner can implement a `connect()` function
- **Disconnections occur due to**
  - `signal.disconnect()` `connection.disconnect()`
  - Tracked object destroyed
  - Signal destroyed

Advanced Topics

- **Slot return values**
  - A slot function may return a value
  - The result of firing a signal is a pointer to the *last* value
  - A signal can define a custom function 'combiner' to process all those return values
- **Postconstructors and predestructors**
  - An object in a `shared_ptr` cannot setup its own tracking in a ctor
  - `deconstruct()`
- **Accessing connection in the slot**
  - `connect_extended`
- **Bonus: notifications on property changes a.k.a. `INotifyPropertyChanged` (warning: compiler extension!)**

Summary

- **`#include <boost/signals2.hpp>`**
- **Create a `signal<T>`**
- **Create one or more slots that match the signature of T**
- **Use `connect()` to wire up**
  - Use the extra parameter for priority
  - Use `shared_ptr` and `slot_type` enable tracking to automatically close the connection when objects die
- **Use `shared_connection_block` to temporarily stop sending signals**
- **Use disconnect() on either the signal or the connection object**
  - Wrap with a `scoped_connection` to disconnect after leaving scope