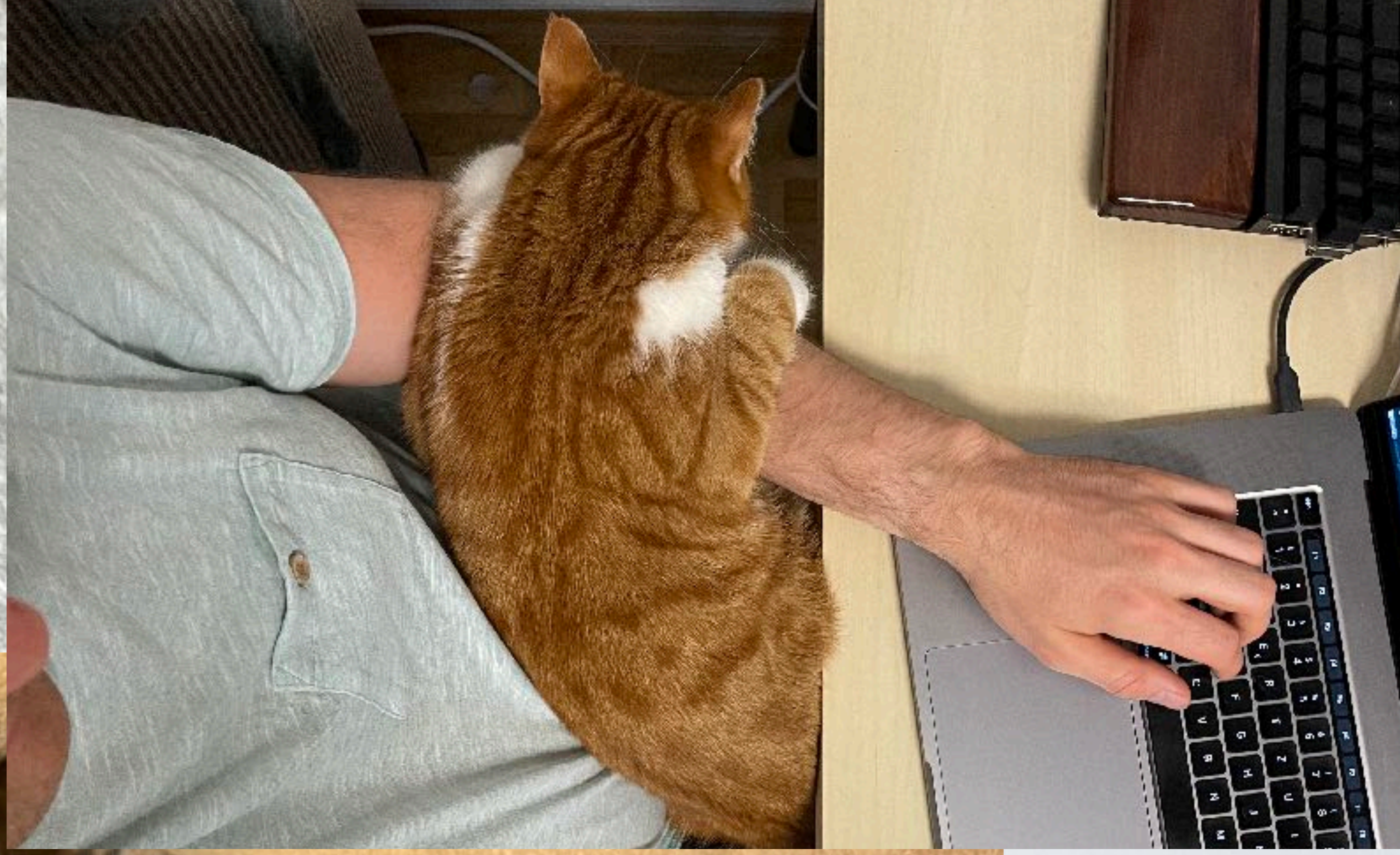
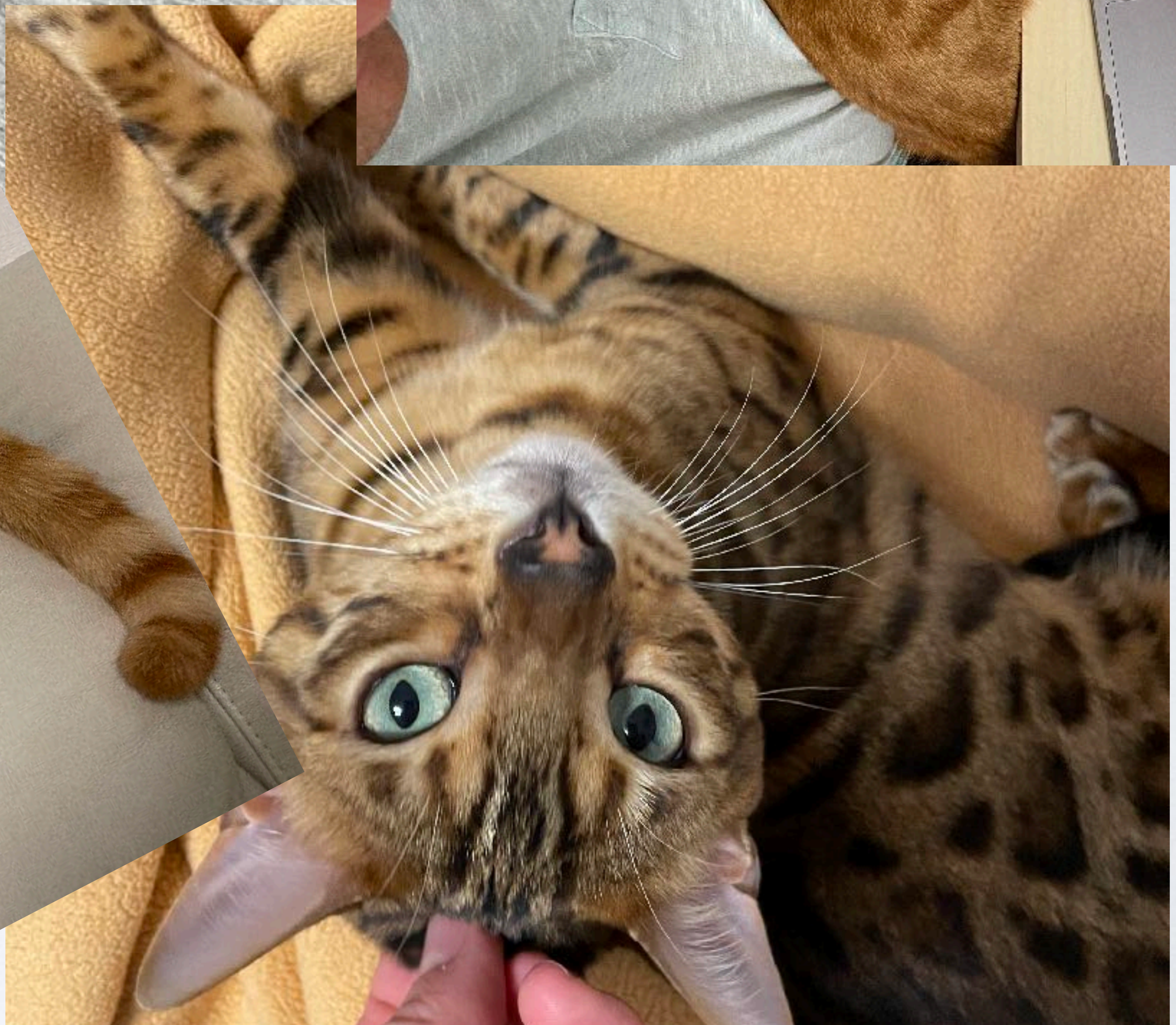
# READING CONFIGURATION VALUES WHEN YOU SHOULD NOT FAIL

From Obfuscated to (Hopefully) Nearly Readable





## THE ENVIRONMENT: PORTING LEGACY CODE EMBEDDED SYSTEMS

## READING CONFIGURATION

- Configuration is accessible by uint64_t id only
- Configuration value is always read using a library
- Reading may fail due to various reasons
- Even if it works the value may be garbage.

First Time Startup
Library Outdated
Software update
Memory Corruption
Manipulation

## THE INTERFACE

```
#define ID_FOR_DISPLAY_WIDTH 1
#define ID_FOR_DISPLAY_HEIGHT 2
#define ID_FOR_REFRESH_RATE 3

int64_t loadFromConfig(uint64_t id, int& error);
```

## THE CONFIGURATION

```
class DisplayConfiguration{
  public:
    DisplayConfiguration();
    uint16_t display_width;  //in pixels, max 4096
    uint16_t display_height; //in pixels, min 480
    uint8_t refresh_rate;    // in 10 milliseconds [10-300]
  private:
    void setInitValues();
    void loadConfig();
}
```

```
DisplayConfiguration::DisplayConfiguration(){
  setInitValues();
  loadConfig();
}

void DisplayConfiguration::setInitValues(){
  display_height = 1024;
  display_width = 1980;
  refresh_rate = 150;
}

...
void DisplayConfiguration::loadConfig(){
  int e;
  display_height = loadFromConfig(ID_FOR_DISPLAY_HEIGHT, e);
  display_width = loadFromConfig(ID_FOR_DISPLAY_WIDTH, e);
}
```

## WHAT I WAS STRUGGLING WITH

- Having to look up at three places
- Again
- And again
- And again…

## EXPRESSIVENESS

## WHAT I WANTED TO HAVE

```
Parameter:
        - std::uint16_t,
        - Load ID_FOR_REFRESH_RATE,
        - Default is 15,
        - Ratio is 10,
        - Max is 300
as t
```

## SO WHY NOT WRITING IT

```
int main() {
  Parameter<
             std::uint16_t,
             Load<ID_FOR_REFRESH_RATE>,
             Default   <15>,
             Ratio   <10,1>,
             Max   <300>
  >  t;
  std::cout << t.Value() << std::endl;
  return 0;
}
```

C++17

```
template <typename T, typename... options>
class Parameter {
  T value_;
 public:
  Parameter() : value_{Load()} {}

  auto Load() -> T {
    std::optional<T> value;
    Apply<T, options...>(value);
    return (value) ? value.value() : T{};
  }

  auto Value() -> T { return value_; }
};
```

C++17

```cpp
template <uint64_t ID>
struct Load {
  template <typename T>
  static void Apply(std::optional<T>& value) {
    if (v.find(ID) != v.end()) {
      value = static_cast<T>(v.at(ID));
    }
  }
};
```

C++17

```
template <typename T, typename Current, typename... Left>
Void Apply(std::optional<T>& value) {

  if constexpr (sizeof...(Left) > 0) {
    Apply<T, Left...>(Current::Apply(value));
  } else {
    Current::Apply(value);
  }

}
```

## REACH OUT

- Discord: m42e#6427
- Mail: matthias@bilger.info
- GitHub: m42e
- https://m42e.de (irregular blog posts even if stated otherwise)
- Twitter: @m42e_de