

---

## Subject: Refactoring Moveable

Posted by [mirek](#) on Fri, 23 Aug 2024 06:52:08 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

In order to make U++ more compatible and future proof, I am changing Moveable mechanisms a bit. U++ will now use C++17 inline template features to simplify Moveable and allow putting "non-U++ guest types" in Vector/BiVector/Index. On the way I hope to fix some other problems (e.g. `auto [a, b] = MakeTuple("x", 1)` does not work yet) and remove all "dangerous" (ok, all possibly undefined behaviour) code, except Moveable, which is de facto standard now anyway (<https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2024/p1144r10.html>).

Development is so far in the branch Core2024, critical part for your kind review:

<https://github.com/ultimatepp/ultimatepp/blob/3638778b2e0e1819622424a70a7f04ef0950741d/uppsrc/Core/Topt.h#L158>

This now works:

```
template <>
inline constexpr bool Upp::is_upp_guest<std::string> = true;

template<> inline hash_t Upp::GetHashValue(const std::string& a)
{
    return memhash(a.data(), a.length());
}
```

```
CONSOLE_APP_MAIN
{
    {
        Vector<std::string> h;
        for(int i = 0; i < 20; i++)
            h << AsString(i).ToStd();
        RDUMP(h);
        Vector<int> rem = { 1, 2, 3 };
        h.Remove(rem);
        RDUMP(h);
        h.RemoveAll([&](int i) { return h[i].back() == '8'; });
        RDUMP(h);
        Vector<std::string> n = { "21", "22", "23" };
        h.Insert(2, n);
        RDUMP(h);
        h.Insert(2, pick(n));
        RDUMP(h);
        h.Remove(2, 3);
        RDUMP(h);
    }
}
```

```
{  
  Index<std::string> x { "one", "two", "three" };  
  RDUMP(x);  
  RDUMP(x.Find("two"));  
}  
}
```

(This works legally, using `std::move` instead of `memmove/memcpy` for `std::string`).

---