
Subject: Re: NEW: generic Toupel grouper
Posted by [mirek](#) on Tue, 31 Aug 2010 11:29:26 GMT
[View Forum Message](#) <> [Reply to Message](#)

dolik.rce wrote on Mon, 30 August 2010 17:13I agree with the a,b,c,d,e... too. It is nice and simple and fast to write. One more question is how many values should be supported. I personally think that a-f should be about enough...

Honza

IMO, 4 is more than enough....

Here is what I got after bit of experimenting:

```
template <typename A, typename B>
struct Tuple2 {
    union {
        A a;
        A key;
    };
    union {
        B b;
        B value;
    };

    int Compare(const Tuple2& x) const    { return CombineCompare(a, x.a)(b, x.b); }
    bool operator==(const Tuple2& x) const { return Compare(x) == 0; }
    bool operator!=(const Tuple2& x) const { return Compare(x) != 0; }
    bool operator<=(const Tuple2& x) const { return Compare(x) <= 0; }
    bool operator>=(const Tuple2& x) const { return Compare(x) >= 0; }
    bool operator<(const Tuple2& x) const  { return Compare(x) != 0; }
    bool operator>(const Tuple2& x) const  { return Compare(x) != 0; }

    unsigned GetHashValue() const        { return CombineHash(a, b); }
};

template <typename A, typename B>
inline Tuple2<A, B> MakeTuple(const A& a, const B& b)
{
    Tuple2<A, B> r;
    r.a = a;
    r.b = b;
    return r;
}

template <typename A, typename B, typename C>
struct Tuple3 {
```

```

union {
    A a;
    A key;
};
union {
    B b;
    B value;
};
union {
    C c;
    C value1;
};

int Compare(const Tuple3& x) const    { return CombineCompare(a, x.a)(b, x.b)(c, x.c); }
bool operator==(const Tuple3& x) const { return Compare(x) == 0; }
bool operator!=(const Tuple3& x) const { return Compare(x) != 0; }
bool operator<=(const Tuple3& x) const { return Compare(x) <= 0; }
bool operator>=(const Tuple3& x) const { return Compare(x) >= 0; }
bool operator<(const Tuple3& x) const  { return Compare(x) != 0; }
bool operator>(const Tuple3& x) const  { return Compare(x) != 0; }

unsigned GetHashValue() const        { return CombineHash(a, b, c); }
};

template <typename A, typename B, typename C>
inline Tuple3<A, B, C> MakeTuple(const A& a, const B& b, const C& c)
{
    Tuple3<A, B, C> r;
    r.a = a;
    r.b = b;
    r.c = c;
    return r;
}

template <typename A, typename B, typename C, typename D>
struct Tuple4 {
    union {
        A a;
        A key;
    };
    union {
        B b;
        B value;
    };
    union {
        C c;
        C value1;
    };
};

```

```

union {
    D d;
    D value2;
};

int Compare(const Tuple4& x) const    { return CombineCompare(a, x.a)(b, x.b)(c, x.c)(d, x.d); }
bool operator==(const Tuple4& x) const { return Compare(x) == 0; }
bool operator!=(const Tuple4& x) const { return Compare(x) != 0; }
bool operator<=(const Tuple4& x) const { return Compare(x) <= 0; }
bool operator>=(const Tuple4& x) const { return Compare(x) >= 0; }
bool operator<(const Tuple4& x) const  { return Compare(x) != 0; }
bool operator>(const Tuple4& x) const  { return Compare(x) != 0; }

unsigned GetHashValue() const        { return CombineHash(a, b, c, d); }
};

template <typename A, typename B, typename C, typename D>
inline Tuple4<A, B, C, D> MakeTuple(const A& a, const B& b, const C& c, const D& d)
{
    Tuple4<A, B, C, D> r;
    r.a = a;
    r.b = b;
    r.c = c;
    r.d = d;
    return r;
}

```
