

---

Subject: Ptr improve

Posted by [tojocky](#) on Mon, 16 May 2011 12:07:58 GMT

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

---

Hello Mirek,

I propose to improve a little Ptr class by add autodelete in PteBase::Prec and Autodelete in PteBase:

in h file:

```
template <class T> class Ptr;
```

```
class PteBase {
```

```
protected:
```

```
    struct Prec {
```

```
        PteBase *ptr;
```

```
        Atomic  n;
```

```
        bool autodelete;
```

```
    };
```

```
    volatile Prec *prec;
```

```
    Prec          *PtrAdd();
```

```
    void Autodelete();
```

```
    static void   PtrRelease(Prec *prec);
```

```
    static Prec   *PtrAdd(const Uuid& uuid);
```

```
    static void Lock();
```

```
    static void Unlock();
```

```
    PteBase();
```

```
    ~PteBase();
```

```
    friend class PtrBase;
```

```
};
```

```
class PtrBase {
```

```
protected:
```

```
    PteBase::Prec *prec;
```

```
    void Set(PteBase *p);
```

```
    virtual void Release();
```

```
    void Assign(PteBase *p);
```

```
public:
```

```
    virtual ~PtrBase();
```

```
};
```

```
template <class T>
```

```
class Pte : public PteBase {
```

```
    friend class Ptr<T>;
```

```

};

template <class T>
class PteAuto : public PteBase {
    friend class Ptr<T>;
public:
    PteAuto(){PteBase::Autodelete();}
};

template <class T>
class Ptr : public PtrBase, Moveable< Ptr<T> > {
    T *Get() const                { return prec&&prec->ptr ? static_cast<T *>(prec->ptr) : NULL; }
protected:
    void Release();
    void Autodelete();
public:
    T *operator->() const          { return Get(); }
    T *operator~() const          { return Get(); }
    operator T*() const           { return Get(); }

    Ptr& operator=(T *ptr)         { Assign(ptr); return *this; }
    Ptr& operator=(const Ptr& ptr) { Assign(ptr.Get()); return *this; }

    Ptr()                         { prec = NULL; }
    Ptr(T *ptr)                   { Set(ptr); }
    Ptr(const Ptr& ptr)            { Set(ptr.Get()); }
    ~Ptr()                        { Autodelete(); }
    String ToString() const;

    friend bool operator==(const Ptr& a, const T *b) { return a.Get() == b; }
    friend bool operator==(const T *a, const Ptr& b) { return a == b.Get(); }
    friend bool operator==(const Ptr& a, const Ptr& b) { return a.prec == b.prec; }

    friend bool operator==(const Ptr& a, T *b)       { return a.Get() == b; }
    friend bool operator==(T *a, const Ptr& b)       { return a == b.Get(); }

    friend bool operator!=(const Ptr& a, const T *b) { return a.Get() != b; }
    friend bool operator!=(const T *a, const Ptr& b) { return a != b.Get(); }
    friend bool operator!=(const Ptr& a, const Ptr& b) { return a.prec != b.prec; }

    friend bool operator!=(const Ptr& a, T *b)       { return a.Get() != b; }
    friend bool operator!=(T *a, const Ptr& b)       { return a != b.Get(); }
};

template<class T>
void Ptr<T>::Release(){
    Autodelete();
    PtrBase::Release();
}

```

```

}

template<class T>
void Ptr<T>::Autodelete(){
    PteBase::Lock();
    if(prec&&prec->autodelete&&prec->ptr&&prec->n < 2){
        delete (T*)(prec->ptr);
        prec->ptr = NULL;
    }
    PteBase::Unlock();
}

template <class T>
String Ptr<T>::ToString() const{
    return prec&&prec->ptr ? FormatPtr(Get()) : String("0x0");
}

```

in cpp:

```

#include "Core.h"

NAMESPACE_UPP

static StaticCriticalSection sPteLock;

void PteBase::Lock(){
    sPteLock.Enter();
}

void PteBase::Unlock(){
    sPteLock.Leave();
}

PteBase::Prec *PteBase::PtrAdd(){
    sPteLock.Enter();
    if(prec) {
        ++prec->n;
        sPteLock.Leave();
    }
    else {
        sPteLock.Leave();
        prec = new Prec;
        prec->n = 1;
        prec->ptr = this;
        prec->autodelete = false;
    }
    return const_cast<Prec *>(prec);
}

```

```

void PteBase::PtrRelease(Prec *prec){
    CriticalSection::Lock __(sPteLock);
    if(prec && --prec->n == 0){
        if(prec->ptr){
            prec->ptr->prec = NULL;
        }
        delete prec;
        prec = NULL;
    }
}

```

```

void PteBase::Autodelete(){
    if(!prec){
        prec = new Prec;
        prec->n = 0;
        prec->ptr = this;
    }
    prec->autodelete = true;
}

```

```

PteBase::PteBase(){
    prec = NULL;
}

```

```

PteBase::~~PteBase(){
    CriticalSection::Lock __(sPteLock);
    if(prec)
        prec->ptr = NULL;
}

```

```

void PtrBase::Release(){
    PteBase::PtrRelease(prec);
}

```

```

void PtrBase::Set(PteBase *p){
    prec = p ? p->PtrAdd() : NULL;
}

```

```

void PtrBase::Assign(PteBase *p){
    Release();
    Set(p);
}

```

```

PtrBase::~~PtrBase(){
    Release();
}

```

END\_UPP\_NAMESPACE

and a little test:

```
#include <Core/Core.h>
```

```
using namespace Upp;
```

```
struct Foo : PteAuto<Foo> {  
    String text;  
};
```

```
Foo* factory(){  
    return new Foo;  
}
```

```
CONSOLE_APP_MAIN
```

```
{  
    Ptr<Foo> ptr;  
    {  
        Ptr<Foo> ptr1 = new Foo;  
        ptr1->text = "Text";  
        ptr = ptr1;  
        Cout() << (void*)~ptr << " -> " << ptr->text << "\n";  
    }  
    Cout() << "-----\n";  
    Cout() << (void*)~ptr << "\n";  
    ptr = factory();  
    ptr = factory();  
}
```

To use this PteAuto you need to know exactly that classes which use this class will automatically deleted.

Any comments are welcome!

PS: changed the realization to resolve with destructor of T.

---