```
// IPtr - Smart Interface Pointer
//   Use:  IPtr<IX, &IID_IX> spIX ;
//   Do not use with IUnknown; IPtr<IUnknown, &IID_IUnknown>
//     will not compile.  Instead, use IUnknownPtr.
template <class T, const IID* piid> class IPtr
{
public:
        // Constructors
        IPtr()
        {
                m_pI = NULL ;
        }

        IPtr(T* lp)
        {
                m_pI = lp ;
                if ( m_pI != NULL)
                {
                        m_pI->AddRef() ;
                }
        }

        IPtr(IUnknown* pI)
        {
                m_pI = NULL ;
                if (pI != NULL)
                {
                        pI->QueryInterface(*piid, (void **)&m_pI) ;
                }
        }

        // Destructor
        ~IPtr()
        {
                Release() ;
        }

        // Reset
        void Release()
        {
                if (m_pI != NULL)
                {
                        T* pOld = m_pI ;
                        m_pI = NULL ;
                        pOld->Release() ;
                }
        }

        // Conversion
        operator T*() { return m_pI ;}

        // Pointer operations
        T& operator*()  { assert(m_pI != NULL) ; return *m_pI ;}
        T** operator&() { assert(m_pI == NULL) ; return &m_pI ;}
        T* operator->() { assert(m_pI != NULL) ; return m_pI ;}

        // Assignment from the same interface
        T* operator=(T* pI)
        {
                if (m_pI != pI)
                {
                        IUnknown* pOld = m_pI ;    // Save current value.
                        m_pI = pI ;                // Assign new value.
                        if (m_pI != NULL)
                        {
                                m_pI->AddRef() ;
                        }
                        if (pOld != NULL)
                        {
                                pOld->Release() ;  // Release the old interf
                        }
                }
                return m_pI ;
        }

        // Assignment from another interface
        T* operator=(IUnknown* pI)
        {
                IUnknown* pOld = m_pI ;       // Save current value.
                m_pI == NULL ;

                // Query for correct interface.
                if (pI != NULL)
                {
                        HRESULT hr = pI->QueryInterface(*piid,
                                                        void**)&m_pI) ;
                        assert(SUCCEEDED(hr) && (m_pI != NULL)) ;
                }

                if (pOld != NULL)
                {
                        pOld->Release() ;         // Release old pointer.
                }
                return m_pI ;
        }

        // Boolean functions
        BOOL operator!() { return (m_pI == NULL) ? TRUE : FALSE ;}

        operator BOOL() const { return (m_pI != NULL) ? TRUE : FALSE ; }

        // GUID
        const IID& iid() { return *piid ;}

private:
        // Pointer variable
        T* m_pI ;
} ;
```

opracowanie: Jarosław Francik