
Subject: Re: Problem - GridCtrl::Get(int,int) with DropTree

Posted by [ktj9](#) on Mon, 31 May 2010 22:18:24 GMT

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

As I said, the problem could be solved by changing GridCtrl source (using dynamic casting to peek DropTree type then do special setting and getting). But the is certainly not a good general solution.

I am trying another solution: define a new Value type that holds both String representation and key/cursor information; derive from DropTree, and the subclass handles the new type in setter and getter. It turns out to be a big exercise of customize Value type. Though Value is supposed to be easy to use, but extending/customizing it is not very fun, at least in this situation. Though I am approaching solving the problem, there are still things are missing that I don't understand. For example, I have to create dummy member functions (see codes below) that basically do nothing but just let templates compile.

Any way, here is my current solution: instead of using DropTree directly, use DropTreeForGridCtrl defined below:

```
class StringWithKey {
    String _s;
    int _key;
public:
    StringWithKey(){}
    StringWithKey(const String& s, int k)
        : _s(s), _key(k)
    {}
    virtual ~StringWithKey(){}
    StringWithKey& operator=(const StringWithKey& rhs)
    {
        if (&rhs==this)
            return *this;
        _s=rhs._s;
        _key=rhs._key;
        return *this;
    }
    StringWithKey& operator=(const Value& rhs)
    {
        return *this;
    }
    StringWithKey& operator<=<=(const StringWithKey& rhs)
    {
        return this->operator=(rhs);
    }
    bool operator==(const StringWithKey& rhs) const
    {
```

```

    return rhs._key==_key && rhs._s==_s;
}
virtual bool    IsEqual(const Value::Void *p)
{
    const StringWithKey* rhs=dynamic_cast<const StringWithKey*>(p);
    if (!rhs)
        return false;
    return operator==(rhs);
}
int  GetKey() const
{
    return _key;
}
StringWithKey& SetKey(int key)
{
    _key=key;
    return *this;
}
virtual String AsString() const
{
    return _s;
}
virtual String ToString() const
{
    return _s;
}
virtual dword   GetType() const
{
    return STRING_V;
}
virtual bool    IsNull() const
{
    return false;
}
bool IsNullInstance() const
{
    return false;
}
void Serialize(Stream& s)
{
}
unsigned GetHashValue() const
{
    return 0;
}
};

```

```

class DropTreeForGridCtrl : public DropTree {

```

```

mutable StringWithKey _value;
public:
DropTreeForGridCtrl(){}
virtual ~DropTreeForGridCtrl(){}
virtual void SetData(const Value& data)
{
    if (!dynamic_cast<const RichValueRep<StringWithKey>*>(data.GetVoidPtr()))
        return;
    StringWithKey t = dynamic_cast<const
RichValueRep<StringWithKey>*>(data.GetVoidPtr())->Get();
    _value=t;
    TreeObject().SetCursor(_value.GetKey());
}
virtual Value GetData() const
{
    _value = StringWithKey(
        const_cast<DropTreeForGridCtrl&>(*this).TreeObject().GetData(),
        const_cast<DropTreeForGridCtrl&>(*this).TreeObject().GetCursor());
    Value ret(new RichValueRep<StringWithKey>(_value));
    return ret;
}
};

```
