Subject: Re: Fixes to Array::Create & Vector::Create
Posted by Novo on Fri, 15 Dec 2017 16:40:49 GMT
View Forum Message <> Reply to Message

Sorry, my bad. The code should look like below.

```
 template<class TT, class... Args>
 TT& Create(Args&&... args)         { TT *q = new TT(std::forward<Args>(args)...); Add(q); return
*q; }

struct Test {
 Test(const Vector<int>& a, int b) : a(a, 0), b(b) {}
 Test(Vector<int>&& a, int b) : a(pick(a)), b(b) {}

 Vector<int> a;
 int         b;
};

CONSOLE_APP_MAIN
{
 Array<Test> h;
 Vector<int> v;
 v.Add(12);
 // Copy-constructor
 h.Create<Test>(v, 22);
 DDUMP(v.GetCount());
 // Move-constructor
 h.Create<Test>(pick(v), 22);
 DDUMP(v.GetCount());
 v.Add(21);
 h.Create<Test>(Vector<int>(v, 0), 22);
 DDUMP(v.GetCount());
}
```

I recompiled TheIDE with this change and it works perfectly for me.

How it works.
type&& is called a universal/perfect/forwarding reference and it can be ether type&& or type& depending on arguments. This is why your method Create(const Args&... args) was redundant. Create(Args&&... args) is a better match.
The problem was that I was using pick, which unconditionally converts type to an rvalue, instead of std::forward, which preservers type (an lvalue will stay the lvalue).