Subject: Re: Should the pick semantics be changed?
Posted by Lance on Wed, 26 Mar 2014 01:48:11 GMT
View Forum Message <> Reply to Message

Quote:
there is an inconsistency in g++: when you write "new C(getC());" then on the heap the C object is initialized with r-value, it's syntactic sugar for "new C(move(getC));". however, if inside getC() you write "C o; createCBasedOnType(type,o); return o;" then this isn't syntactic sugar for "C o; createCBasedOnType(type,o); return move(o);". in the first case the syntactic sugar is rationalized by the impermanence of the C object getC() sends to the initializer. but also the object o which is the return-value of getC() is impermanent, it gets destroyed immediately after the return-value has been created. it is an inconsistency that both cases aren't treated the same in terms of syntactic sugar! it's not really a bug, but it's counterintuitive.

what g++ does is exactly what's required by the c++11 standard and hence will be done by compliant vc++ or other c++.

First case, getC() returns a tempoary (unnamed_, which is a r-value reference. while in the second case, o is named, std::move() utility or its equivalent is required to convert it into a r-value reference.

By the way, it has nothing to do with where it's created on the heap, it has to do with whether you are constructing with a temporary or a named a varaible, eg

```
struct C{

C(const C& c){}
C(C&& c){}

};

C getC()
{
   C c;
   return std::move(c);
}

int test()
{
   C c=getC(); // the move constructor will be called;
   C*p=new C(getC()); // same move constructor;

   C d=c; // copy ctor will be called;


   C e=std::move(c); // with named variable, std::move or
         // equivalent is required to invoke the move ctor
```

```
}
```