Subject: Re: Should the pick semantics be changed?
Posted by Lance on Sun, 23 Mar 2014 21:02:27 GMT
View Forum Message <> Reply to Message

piotr5 wrote on Sun, 09 March 2014 08:28well, to formalize a bit more concretely what I said, I believe return statement should at compile-time distinguish between return values local to the function which are not static, and all the return-values which remain in scope after the end of this function respectively could get into scope again (like static values or private members and such). and then based on this distinction the actual return-value should either be constructed with implicit cast to r-value or with the persistent const-l-value-reference constructor.

Not sure if I get you correctly but I think it's logical to put the burden on programmers as is. The two return types are different (and binary incompitable): one is an object and the other is a reference to object. Let's say C is a moveable class. There are some global/static objects of C which we can choose from one only from base on input parameter, then it's logical for the programmer the let the reference be the return type.

```
 /*const*/ C& getC(int type);
```

Now suppose we need to return some composed(not presently existing) C for some special input types(e.g., when type<100, there are corresponding existing global/static C objects, when type>=100, one has to be composed accordingly), the most economically way is to return a temporary C object if adding a new function to handle the case when type>=100 is not desirable.

```
C getC(int type);
```

Now if you declare the function prototype using the second way, any reasonable compiler would not be able to change it to return reference (the first) even if it's smart enough to detect that inside you implementation for getC actually involves only global/static C objects.

So a programmer has to make this decision, the compiler has to respect the programmer even if doing it otherwise is more efficient.

However, in the case a local object is returned, compiler may take the burden and save the programmer some extra key stroke., eg

```
C getC(int type)
{
   if( (unsigned)type<100 )
      return globalCs[type]; // return reference here is
                             // impossible as it will make
                             // the subsequent code impossible

   C c;
   createCBasedOnType(type, c);
```

```
    return c; // instead of
            // return std::move(c);
            // as required by current standard to take
            // advantage of the supposedly more efficient
            // move semantic
}
```