Subject: Value: no deep copy support?
Posted by kohait00 on Thu, 02 Sep 2010 13:58:10 GMT
View Forum Message <> Reply to Message

hi all,

dealing with Value i noticed that it has no possibility to do a clone or DeepCopyNew of the referenced content.

this is surely by design, but what were the concepts leading to this?

i know there is workarounds for this but having it directly supported would be great. a value is Movable, DeepCopyOption would be great too.

workaround:

Value v = int(123); //initial stuff, put in there, as a copy of this int i.e.

Value v2 = v1; //would internally reference the same instance of int

Value v3 = (int)v1; //this creates a copy from the internal

http://www.ultimatepp.org/srcdoc$Core$UserValue$en-us.html

i'd be great to spare this out and use the pick'able semantic from containers..

Value v = (int)123;
Value v2(v,0); //indicate to use the deep copy

Value v3;
vs <<= v; //same

do i misunderstand somehow the concept of Value maybe? (i know that it's a referece object internally managing a 'shared' content)

EDIT: maybe it's a good idea to collect all things related to Value handling. they are spread in manual and short tutorials etc.

http://www.ultimatepp.org/reference$Value.html
http://www.ultimatepp.org/srcdoc$Core$CoreTutorial$en-us.htm l
http://www.ultimatepp.org/srcdoc$Core$UserValue$en-us.html
http://www.ultimatepp.org/src$Draw$Display$en-us.html
http://www.ultimatepp.org/www$uppweb$overview$en-us.html
http://www.ultimatepp.org/src$Core$Convert$en-us.html

Subject: Re: Value: no deep copy support?
Posted by mirek on Mon, 06 Sep 2010 08:38:50 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Thu, 02 September 2010 09:58hi all,

dealing with Value i noticed that it has no possibility to do a clone or DeepCopyNew of the referenced content.

this is surely by design, but what were the concepts leading to this?

Huh? Value is supposed to work "value-like" at all times, just like fundamental types (say 'int'). Or, to say it better, just like String or Time.

Forget about "clones" or "deep copies", these concepts do not apply here.

(And yes, internally Value implementation now uses reference counting - note however that this might change in future to some more sophisticated pattern - just like it did with String).

Subject: Re: Value: no deep copy support?
Posted by mirek on Mon, 06 Sep 2010 08:39:35 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Thu, 02 September 2010 09:58
http://www.ultimatepp.org/reference$Value.html
http://www.ultimatepp.org/srcdoc$Core$CoreTutorial$en-us.htm l
http://www.ultimatepp.org/srcdoc$Core$UserValue$en-us.html
http://www.ultimatepp.org/src$Draw$Display$en-us.html
http://www.ultimatepp.org/www$uppweb$overview$en-us.html
http://www.ultimatepp.org/src$Core$Convert$en-us.html

You are welcome to do so

Reference Value documentation is heavily lacking as well...

Subject: Re: Value: no deep copy support?
Posted by kohait00 on Wed, 20 Oct 2010 14:08:10 GMT
View Forum Message <> Reply to Message

there are almost no infos on classes like ValueArray / ValueMap. I ended up trying to code sth. similar, before relizing that it's already there..but i actually dont know exactly how to use them..what the design rule was.

and: as far as got to play with it in persistancy: there seems to be a BUG...
see
http://www.ultimatepp.org/forum/index.php?t=msg&goto=294 37&#msg_29437

Subject: Re: Value: no deep copy support?
Posted by kohait00 on Wed, 20 Oct 2010 14:55:58 GMT
View Forum Message <> Reply to Message

back to the topic:

offering a somehow optional way to be able to deepcopy the content of a Value would make it an appealing refcount based container for polymorphic stuff as well. Array<T> still needs some info about type.
Value doesnt.

i dont mean copy-on-write or something, which would be implicit, so the user does not get an idea of behaviour. he still can expect Value to be one (of many) references to the content.

but wanting to internally clone it without knowing type or anything about it would be great as well. enabling the user to 'edit' later 2 distinct copies in an abstract handler.

it could maybe rely on DeepCopyNew somehow. but this might need to introduce a Copy function in the Value::Void interface, returning Void*.

```
virtual Void* Copy() const { return new Void(); }
```

the RawValueRep<T> would add sth like:

```
virtual RawValueRep* Copy() const { return new RawValueRep(*this); }
```

```
//the value could then be MoveableAndDeepCopyOption

Value(const Value& v, int)
{
 ptr = v.ptr->Copy();
 //ptr->Retain(); //we are first owner
}
```

this would enable the approach described above

```
Value v = (int)123;
Value v2(v,0); //indicate to use the deep copy to duplicate the values

//v and v2 are now unrelated anymore..can be edited/overwritten individually
```

//using const_cast..clone has been produced without knowing content type.

Value v3;
vs <<= v; //same as with constructor, more intuitive, as from Containers :)

these are only very basic ideas, maybe it is of interest. i'll try to play with it a bit..maybe i can find sth usefull.

Subject: Re: Value: no deep copy support?
Posted by mirek on Wed, 20 Oct 2010 21:16:59 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Wed, 20 October 2010 10:55back to the topic:

offering a somehow optional way to be able to deepcopy the content of a Value would make it an appealing refcount based container for polymorphic stuff as well. Array<T> still needs some info about type.
Value doesnt.

i dont mean copy-on-write or something, which would be implicit, so the user does not get an idea of behaviour. he still can expect Value to be one (of many) references to the content.

but wanting to internally clone it without knowing type or anything about it would be great as well. enabling the user to 'edit' later 2 distinct copies in an abstract handler.

Nonsense. You can NEVER 'edit' Value.

Quote:
it could maybe rely on DeepCopyNew somehow. but this might need to introduce a Copy function in the Value::Void interface, returning Void*.

virtual Void* Copy() const { return new Void(); }

the RawValueRep<T> would add sth like:

virtual RawValueRep* Copy() const { return new RawValueRep(*this); }

//the value could then be MoveableAndDeepCopyOption

```
Value(const Value& v, int)
{
 ptr = v.ptr->Copy();
 //ptr->Retain(); //we are first owner
```

}

this would enable the approach described above

Value v = (int)123;
Value v2(v,0); //indicate to use the deep copy to duplicate the values

//v and v2 are now unrelated anymore..can be edited/overwritten individually
//using const_cast..clone has been produced without knowing content type.

Value v3;
vs <<= v; //same as with constructor, more intuitive, as from Containers :)

these are only very basic ideas, maybe it is of interest. i'll try to play with it a bit..maybe i can find sth usefull.

Sorry, but this is just misunderstanding. There is zero value in deep copy for Value.

Please think about Value like it is an 'int' - with "assignment" time always O(1).

That implies Value is principally immutable - you can never change what is inside. Only assign another value.

Subject: Re: Value: no deep copy support?
Posted by kohait00 on Thu, 21 Oct 2010 06:47:13 GMT
View Forum Message <> Reply to Message

thanks mirek, got to rethink the concept.
i'm a more or less a freshy in all that, programming idioms and tricks in c++. graduated not to a sowftware engeneer, but technical and hardware but happened to begin to work in software engeneering. where i need to teach my self a lot.
thanks upp speeds up all of that. thanks

EDIT: just a question aside:
why is then ValueArray there, which is sort of a container? and why the possibility to 'hide' own custom data inside Value, using RawValue or RichValue..

isnt Value meant to be a ref count based reference to some abstract data (not only the intrinsic ones), and to even be able to edit it as presented in one of the Value tutorials, using const_cast? i think the content is meant to be editable, under well known conditions..

Subject: Re: Value: no deep copy support?

Posted by mirek on Thu, 21 Oct 2010 17:06:23 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Thu, 21 October 2010 02:47thanks mirek, got to rethink the concept.
i'm a more or less a freshy in all that, programming idioms and tricks in c++. graduated not to a sowftware engeneer, but technical and hardware but happened to begin to work in software engeneering. where i need to teach my self a lot.
thanks upp speeds up all of that. thanks

EDIT: just a question aside:
why is then ValueArray there, which is sort of a container? and why the possibility to 'hide' own custom data inside Value, using RawValue or RichValue..

Why not?

Quote:
isnt Value meant to be a ref count based reference to some

That is just implementation detail. In fact, I was thinking that certain types will be stored differently.

Quote:
abstract data (not only the intrinsic ones), and to even be able to edit it as presented in one of the Value tutorials, using const_cast?

Ops, where?

Surely, you can use const_cast brute force there, but that is very very bad idea. If this is in any example, it should be removed quickly.

Subject: Re: Value: no deep copy support?
Posted by kohait00 on Thu, 21 Oct 2010 18:17:08 GMT
View Forum Message <> Reply to Message

it was sort of retorical question anyway. RawValue / RichValue / ValueArray in any case do represent data 'container' and not only the information it has. and as a container one wants to have access modify access to the container.. (int 123 is information, int& 123 references a *container* which happens to have 123 as information).
it's difficult to explain actually.

actually, i dont see much of the point to have Value reference anything else than the intrinsic datatypes, if they are immutable. maybe i need some time to wrap my mind around Value fully

maybe i need to think of it as a reference to a bit of very volatile readonly data..that is created

somewhere in the dust, and if it ceases to be of importance it becomes dust again..
you actually never need to reference the same data packet inside to rely on its information. if its content (or better the information it represents) is about to be altered, this info is created. so Value is not about the place to store sth. but about to represent some information, speeded up by ref count. sorry, trying to understand by describing for the purpose of referencing a container maybe one should use Ptr<T>..

you asked about the const thing:

http://www.ultimatepp.org/srcdoc$Core$UserValue$en-us.html
Quote:
Note that the function ValueTo returns a constant reference. This is consistent with default Value behaviour, according to which data once put into Value should never change afterwards. This is necessary because, upon copying, multiple variables of type Value can hold the same data "packet" and changing its contents would affect all these copies, which is normally undesirable. If you break the const-ness using a mutable member or by a const_cast, you should not forget that.

this actually adds to understand Value as a container for sth. ("packet", "contents").

BTW: maybe you could help me with the ValueArray understanding in another thread here in Core++

Subject: Re: Value: no deep copy support?
Posted by mirek on Thu, 21 Oct 2010 21:23:04 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Thu, 21 October 2010 14:17it was sort of retorical question anyway. RawValue / RichValue / ValueArray in any case do represent data 'container' and not only the information it has. and as a container one wants to have access modify access to the container.. (int 123 is information, int& 123 references a *container* which happens to have 123 as information).
it's difficult to explain actually.

actually, i dont see much of the point to have Value reference anything else than the intrinsic datatypes, if they are immutable. maybe i need some time to wrap my mind around Value fully

maybe i need to think of it as a reference to a bit of very volatile readonly data..that is created somewhere in the dust, and if it ceases to be of importance it becomes dust again..
you actually never need to reference the same data packet inside to rely on its information. if its content (or better the information it represents) is about to be altered, this info is created. so Value is not about the place to store sth. but about to represent some information, speeded up by ref count. sorry, trying to understand by describing for the purpose of referencing a container maybe one should use Ptr<T>..

you asked about the const thing:

http://www.ultimatepp.org/srcdoc$Core$UserValue$en-us.html

Quote:
Note that the function ValueTo returns a constant reference. This is consistent with default Value behaviour, according to which data once put into Value should never change afterwards. This is necessary because, upon copying, multiple variables of type Value can hold the same data "packet" and changing its contents would affect all these copies, which is normally undesirable. If you break the const-ness using a mutable member or by a const_cast, you should not forget that.

this actually adds to understand Value as a container for sth. ("packet", "contents").

Sorry, but what I read in that paragraph is "do not do this. If you insist to do, be prepared to shot your leg off". And it is more about mutable members used to caching information than anything else.

Quote:
BTW: maybe you could help me with the ValueArray understanding in another thread here in Core++

I guess you are using bad approach here. Why do not you just check for what Value, ValueArray and ValueMap are actually used?

Here are some good examples:

http://www.ultimatepp.org/reference$ArrayCtrl$en-us.html
http://www.ultimatepp.org/reference$Display$en-us.html
http://www.ultimatepp.org/reference$SQL_Sqlite3$en-us.html
http://www.ultimatepp.org/reference$Switch$en-us.html
http://www.ultimatepp.org/reference$Chameleon$en-us.html

Subject: Re: Value: no deep copy support?
Posted by kohait00 on Thu, 21 Oct 2010 21:30:26 GMT
View Forum Message <> Reply to Message

thanks, i'll dig them.
