Subject: ValueArray behaviour / inconsistantcy / BUG?
Posted by kohait00 on Wed, 20 Oct 2010 07:33:26 GMT
View Forum Message <> Reply to Message

hi guys, i'm trying to make ValueArray work for me.
i cant quite get it to the point to be persistant, means to serialize and xmlize properly..

maybe one of you familiar with ValueArray could quickly spot the error in my code bellow (if there is any).

the test prog is attached as well as a ready to run package.
thanks a lot in advance.

Xmlize of ValueArray is kinda custom, but seems to work..
Serialize OTOH does not, and i dont exactly know why..
it deserializes but i cant get the ValueArray back from beeing Value.

```
#include <Core/Core.h>

using namespace Upp;

void ValueArrayXmlize(XmlIO xml, Value& v)
{
 if(xml.IsStoring())
 {
  const ValueArray& va = v;
  XmlizeStore(xml, va.Get());
 }
 if(xml.IsLoading())
 {
  ValueArray va;
  Vector<Value>& vv = const_cast<Vector<Value>&>(va.Get());
  ::Xmlize(xml, vv);
  v = va;
 }
}
INITBLOCK { RegisterValueXmlize(GetValueTypeNo<ValueArray>(), &ValueArrayXmlize,
"ValueArray"); }


CONSOLE_APP_MAIN
{
 ValueArray va;
 Vector<Value>& vv = const_cast<Vector<Value>&>(va.Get());
 vv << 123;
 vv << "Hallo";
```

```
 Value v = va;

 RLOG(v);

 StoreAsXMLFile(v);
 StoreToFile(v);

 Value v2;
 LoadFromXMLFile(v2);

 RLOG(v2);

 const ValueArray& va2 = v2;

 const Vector<Value>& vv2 = va2.Get();

 ASSERT(v2.Is<ValueArray>());
 ASSERT(va2.GetCount() == va.GetCount());
 ASSERT(vv2.GetCount() == vv.GetCount());

 for(int i = 0; i < vv2.GetCount(); i++)
  ASSERT(vv2[i] == vv[i]);


 Value v3;
 LoadFromFile(v3);

 RLOG(v3);

 const ValueArray& va3 = v3; //CRASH

 const Vector<Value>& vv3 = va3.Get();

 ASSERT(v3.Is<ValueArray>());
 ASSERT(va3.GetCount() == va.GetCount());
 ASSERT(vv3.GetCount() == vv.GetCount());

 for(int i = 0; i < vv3.GetCount(); i++)
  ASSERT(vv3[i] == vv[i]);
}
```

File Attachments

1) ValueArrayTest.rar, downloaded 228 times

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?

i've been debugging a bit, and am pretty sure thats a bug in ValueArray:

having refactored my Xmlize procedure to

```
#include <Core/Core.h>

using namespace Upp;

void ValueArrayXmlize(XmlIO xml, Value& v)
{
 if(xml.IsStoring())
 {
  const ValueArray& va = v;
  XmlizeStore(xml, va.Get());
 }
 if(xml.IsLoading())
 {
  Vector<Value> vv;
  ::Xmlize(xml, vv);
  v = ValueArray(vv);
 }
}
INITBLOCK { RegisterValueXmlize(GetValueTypeNo<ValueArray>(), &ValueArrayXmlize,
"ValueArray"); }


CONSOLE_APP_MAIN
{
 Vector<Value> vv;
 vv << 123;
 vv << "Hallo";

 ValueArray va(vv); //picks vv, ValueArray::Data contains the Vector, ValueArray ref count
references the VA::Data
 Value v = va; //the Value now additionally references the ValueArray::Data

 RLOG(v);

 ValueArray va_ = v; //now another ValueArray references the same ValueArray::Data
 const Vector<Value>& vv_ = va_.Get();
 DUMPC(vv_);

 StoreAsXMLFile(v); //the pure Value is xmlized with the ValueArray::Data as Void derived
content, not as RichValue
 StoreToFile(v); //the pure Value is serialized with the ValueArray::Data as Void derived content,
not as RichValue
```

```
 Value v2;
 LoadFromXMLFile(v2); //the Value is dexmlized, with help of ValueArray to only create the ::Void
derived Container
 RLOG(v2);

 //some checks
 ASSERT(IsValueArray(v2)); //is OK
 const ValueArray& va2 = v2;
 const Vector<Value>& vv2 = va2.Get();
 ASSERT(va2.GetCount() == va.GetCount());
 for(int i = 0; i < vv2.GetCount(); i++)
  ASSERT(vv2[i] == va[i]);

 Value v3;
 LoadFromFile(v3); //is deserilized as RichValue, with ValueArray as content, wich has
ValueArray::Data as its content
 RLOG(v3); //works fine anyway

 ASSERT(IsValueArray(v3)); //still no problem, though not consistent.
 const ValueArray& va3 = v3; //CRASH, here comes the inconsistancy
 const Vector<Value>& vv3 = va3.Get();
 ASSERT(va3.GetCount() == va.GetCount());
 for(int i = 0; i < vv3.GetCount(); i++)
  ASSERT(vv3[i] == va[i]);
}
```

it works as expected, as far as i understand.

now the question is, how is ValueArray supposed to be used:

is it supposed to be used as
1) Content of a RichValueRep? (Value.cpp:142: RichValue<ValueArray>::Register()
2) as a selfsustained Value interface/data container, just same as RawValue / RichValue?

as far as i got it to understand, ValueArray is kind of both. an extended Value implementation and meant to be in container data of a normal value, but offers the ValueArray::Data : Value::Void implementation, which is sort of Value domain.

now converting a ValueArray to a Value takes over the ValueArray::Data, serilizing it properly. deserializing it creates a RichValueRep<ValueArray>, which is inconsistent.

>>> so how is ValueArray supposed to be handled? how to deal with the inconsistancy? i'd suggest to make ValueArray / ValueMap a true Value derive...not a RichValue related one.. this comes closer to the idea of the ::Void derived refcounted ValueArray::Data

here, an updated testcase

File Attachments
1) ValueArrayTest.rar, downloaded 216 times

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by kohait00 on Wed, 20 Oct 2010 14:00:57 GMT
View Forum Message <> Reply to Message

to show a bit more of the inconsistancy:

```
Vector<Value> vv;
vv << 123;
vv << "Hallo";

ValueArray va(vv); //picks vv, ValueArray::Data contains the Vector, ValueArray ref count references the VA::Data
Value v = va; //the Value now additionally references the ValueArray::Data

RLOG(v);

ASSERT(v.Is<ValueArray>()); //OK, but should be ERROR
ASSERT(IsValueArray(v)); //OK, evaluates ::Data Content directly
```

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by mirek on Thu, 21 Oct 2010 20:40:13 GMT
View Forum Message <> Reply to Message

I confirm this as bug, fixing now (it is caused by ValueArray (and ValueMap) optimization, where Value inside holds pointer to ValueArray::Date, not ValueArray as any other type).

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by mirek on Thu, 21 Oct 2010 21:08:43 GMT
View Forum Message <> Reply to Message

Fixed.

BTW, NEVER EVER do stuff like:

```
Vector<Value>& vv = const_cast<Vector<Value>&>(va.Get());
vv << 123;
vv << "Hallo";
```

this will likely crash any code pretty soon.

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by mirek on Thu, 21 Oct 2010 21:12:41 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Wed, 20 October 2010 04:45
now the question is, how is ValueArray supposed to be used:

is it supposed to be used as
1) Content of a RichValueRep? (Value.cpp:142: RichValue<ValueArray>::Register()
2) as a selfsustained Value interface/data container, just same as RawValue / RichValue?

as far as i got it to understand, ValueArray is kind of both. an extended Value implementation and meant to be in container data of a normal value, but offers the ValueArray::Data : Value::Void implementation, which is sort of Value domain.

now converting a ValueArray to a Value takes over the ValueArray::Data, serilizing it properly. deserializing it creates a RichValueRep<ValueArray>, which is inconsistent.

It was only a bug.

When using Value, you should not care about RichValueRep as long as you are not adapting another type to be Value compatible.

ValueArray is simply array of Values with O(1) copy, which is Value compatible. Nothing more, nothing less.

As far as that "ValueArray::Data" issue, it is simply optimization, which implements ValueArray without RichValueRep helper. The bug was that we were not careful enough and Serialization supposed that ValueArray Value compatibility IS done using RichValueRep.

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by kohait00 on Thu, 21 Oct 2010 21:32:50 GMT
View Forum Message <> Reply to Message

thanks mirek..i'll check back with my package whether it works..
BTW: DDUMPS left over in cpp

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?

Posted by mirek on Thu, 21 Oct 2010 22:09:15 GMT
View Forum Message <> Reply to Message

ops, thx

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by kohait00 on Fri, 22 Oct 2010 05:30:43 GMT
View Forum Message <> Reply to Message

tested, bug is gone, thanks

BTW: what about the

```
void ValueArrayXmlize(XmlIO xml, Value& v)
{
 if(xml.IsStoring())
 {
  const ValueArray& va = v;
  XmlizeStore(xml, va.Get());
 }
 if(xml.IsLoading())
 {
  Vector<Value> vv;
  ::Xmlize(xml, vv);
  v = ValueArray(vv);
 }
}
INITBLOCK { RegisterValueXmlize(GetValueTypeNo<ValueArray>(), &ValueArrayXmlize,
"ValueArray"); }
```

does it make sense to provide it in Core?

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by mirek on Fri, 22 Oct 2010 07:53:30 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Fri, 22 October 2010 01:30tested, bug is gone, thanks

BTW: what about the

```
void ValueArrayXmlize(XmlIO xml, Value& v)
{
 if(xml.IsStoring())
 {
  const ValueArray& va = v;
  XmlizeStore(xml, va.Get());
```

```
 }
 if(xml.IsLoading())
 {
  Vector<Value> vv;
  ::Xmlize(xml, vv);
  v = ValueArray(vv);
 }
}
INITBLOCK { RegisterValueXmlize(GetValueTypeNo<ValueArray>(), &ValueArrayXmlize,
"ValueArray"); }
```

does it make sense to provide it in Core?

No.

But what would make sense is:

Xmlize specialization for ValueArray and ValueMap.

Backward compatible change of Xmlize of Value that would handle 'basic known' types (that would include ValueArray) more gently, without binary serialization.

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by kohait00 on Fri, 22 Oct 2010 08:12:26 GMT
View Forum Message <> Reply to Message

i dont know the API too much to know where to do it..
just prepared sth. i think to have understood from your post.
maybe you can do sth about it..

```
NAMESPACE_UPP
template<> void Xmlize(XmlIO xml, ValueArray& v)
{
 if(xml.IsStoring())
 {
  XmlizeStore(xml, v.Get());
 }
 if(xml.IsLoading())
 {
  Vector<Value> vv;
  ::Xmlize(xml, vv);
  v = ValueArray(vv);
 }
}
```

```
END_UPP_NAMESPACE


void ValueArrayXmlize(XmlIO xml, Value& v)
{
 if(xml.IsStoring())
 {
  const ValueArray& va = v;
  XmlizeStore(xml, va);
 }
 if(xml.IsLoading())
 {
  ValueArray va;
  ::Xmlize(xml, va);
  v = va;
 }
}
INITBLOCK { RegisterValueXmlize(GetValueTypeNo<ValueArray>(), &ValueArrayXmlize,
"ValueArray"); }
```

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by kohait00 on Fri, 22 Oct 2010 08:17:23 GMT
View Forum Message <> Reply to Message

it even works with just

```
NAMESPACE_UPP
template<> void Xmlize(XmlIO xml, ValueArray& v)
{
 if(xml.IsStoring())
 {
  XmlizeStore(xml, v.Get());
 }
 if(xml.IsLoading())
 {
  Vector<Value> vv;
  ::Xmlize(xml, vv);
  v = ValueArray(vv);
 }
}
END_UPP_NAMESPACE

REGISTER_VALUE_XMLIZE(ValueArray);
```

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by mirek on Fri, 22 Oct 2010 09:01:42 GMT
View Forum Message <> Reply to Message

..and at least ValueMap too, please

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by kohait00 on Fri, 22 Oct 2010 11:52:35 GMT
View Forum Message <> Reply to Message

ValueMap is sure more complicated. i tried it..realizing btw. that Index and ArrayIndex are not Xmlize'able yet. because they dont export API to access the hash separetely, so one can't cleanly xmlize it, only under assumption that hashes have been created from the objects them selves; nevertheless, code explains more.

and another bug (dunno, could be): XmlizeMap does not Clear() the container at the beginning, like XmlizeContainer does.

NAMESPACE_UPP

```
//old version without preparaton for hashfn awareness
#if 0
template<class T>
void Xmlize(XmlIO xml, Index<T>& data)
{
 if(xml.IsStoring())
 {
  XmlizeStore(xml, data.GetKeys()); //FIXME xmlize with hashfn awareness
 }
 if(xml.IsLoading())
 {
  data.Clear();
  Vector<T> keys;
  ::Xmlize(xml, keys); //FIXME dexmlize with hashfn awareness
  data = Index<T>(keys);
 }
}
#endif

template<class K, class T>
void XmlizeIndex(XmlIO xml, const char *keytag, const char *valuetag, T& data)
{
 if(xml.IsStoring()) {
  for(int i = 0; i < data.GetCount(); i++)
   if(!data.IsUnlinked(i)) {
```

```
    //XmlizeStore(xml.Add(keytag), data.GetKey(i)); //FIXME xmlize with hashfn awareness
    XmlizeStore(xml.Add(valuetag), data[i]);
   }
 }
 else {
  data.Clear();
  int i = 0;
  //while(i < xml->GetCount() - 1 && xml->Node(i).IsTag(keytag) && xml->Node(i +
1).IsTag(valuetag)) {
  while(i < xml->GetCount() && xml->Node(i).IsTag(valuetag)) {
   //K key;
   //Xmlize(xml.At(i++), key); //FIXME dexmlize with hashfn awareness
   K k;
   ::Xmlize(xml.At(i++), data.Add(k));
  }
 }
}

template<class K, class H>
void Xmlize(XmlIO xml, Index<K, H>& data)
{
 XmlizeIndex<K>(xml, "key", "value", data);
}

template<class K, class H>
void Xmlize(XmlIO xml, ArrayIndex<K, H>& data)
{
 XmlizeIndex<K>(xml, "key", "value", data);
}

template<> void Xmlize(XmlIO xml, ValueArray& v)
{
 if(xml.IsStoring())
 {
  XmlizeStore(xml, v.Get());
 }
 if(xml.IsLoading())
 {
  Vector<Value> vv;
  ::Xmlize(xml, vv);
  v = ValueArray(vv);
 }
}

template<> void Xmlize(XmlIO xml, ValueMap& v)
{
 if(xml.IsStoring())
 {
```

```
  XmlizeStore(xml, v.GetKeys());
  XmlizeStore(xml, v.GetValues());
 }
 if(xml.IsLoading())
 {
  Index<Value> vv;
  ::Xmlize(xml, vv);
  ValueArray va;
  ::Xmlize(xml, va);
  ASSERT(vv.GetCount() == va.GetCount());
  ValueMap vm;
  for(int i = 0; i < vv.GetCount(); i++)
   vm.Add(vv[i], va[i]);
 }
}
END_UPP_NAMESPACE
```

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by kohait00 on Mon, 25 Oct 2010 11:59:52 GMT
View Forum Message <> Reply to Message

has there been any other thought on why not Xmlize Index and ArrayIndex?

BTW: dont forget the bugfix
Quote:
and another bug (dunno, could be): XmlizeMap does not Clear() the container at the beginning, like XmlizeContainer does.

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by mirek on Wed, 27 Oct 2010 17:32:41 GMT
View Forum Message <> Reply to Message

Fixed and commited.

Thanks.

Mirek

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by kohait00 on Thu, 28 Oct 2010 05:45:50 GMT
View Forum Message <> Reply to Message

nice addition GetHash

just a quick fix:

Xmlize.cpp:258

```
REGISTER_VALUE_XMLIZE(ValueArray);
REGISTER_VALUE_XMLIZE(ValueMap);
```

//now here comes the changed version for Index xmlize, with hash awareness

Xmlize.h:198

```
template<class K, class T>
void XmlizeIndex(XmlIO xml, const char *keytag, const char *valuetag, T& data)
{
 if(xml.IsStoring()) {
  for(int i = 0; i < data.GetCount(); i++)
   if(!data.IsUnlinked(i)) {
    XmlizeStore(xml.Add(keytag), (int64)data.GetHash(i));
    XmlizeStore(xml.Add(valuetag), data[i]);
   }
 }
 else {
  data.Clear();
  int i = 0;
  while(i < xml->GetCount() - 1 && xml->Node(i).IsTag(keytag) && xml->Node(i +
1).IsTag(valuetag)) {
   int64 hash;
   Xmlize(xml.At(i++), hash);
   K key;
   Xmlize(xml.At(i++), key);
   data.Add(key, (unsigned)hash);
  }
 }
}
```

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by mirek on Thu, 28 Oct 2010 08:10:51 GMT
View Forum Message <> Reply to Message

Well, there is one little issue there and one big issue:

First - you do not need two tags for index, just keytag (or valuetag, but I would stay with key). This one I have fixed.

Second, if we are about to be as correct as to add actual hashes, we should also implement 'Unlinked' more correctly, I mean by adding all unlinked elements too and unlinking them...

Also, hash+unlinked holds for Map too.

My suggestion is to use "unlinkedkey" tag for unlinked items instead of "key".

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by kohait00 on Thu, 28 Oct 2010 16:20:53 GMT
View Forum Message <> Reply to Message

ahh.. i think i got u now.
Maps also have ability to externally specify the hash under which a value should be acessible, besides the key specified

T& Add(const K& k, unsigned hash);

and it neither is serialized in the container..
so no need to serialize it in index as well, or do it in all cases.
well, makes sense..

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by mirek on Fri, 29 Oct 2010 06:37:56 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Thu, 28 October 2010 12:20ahh.. i think i got u now.
Maps also have ability to externally specify the hash under which a value should be acessible, besides the key specified

T& Add(const K& k, unsigned hash);

and it neither is serialized in the container..

Which is not right as well (and it is my fault).

Of course, in practice, it is little difference...

Maybe we can leave it to sleep for now... (We would have to invent some method how to make Serialize backward compatible).

Subject: Re: ValueArray behaviour / inconsistantcy / BUG?
Posted by kohait00 on Fri, 29 Oct 2010 10:13:26 GMT
View Forum Message <> Reply to Message

i agree, no problem..one should be aware of it, when using persistance eitherway, xml or serialize..that when overriding internal hash generators it is not possible to preserve the hashes.