Subject: A [TreeCtrl] bug
Posted by huanghuan on Mon, 03 Mar 2008 06:56:58 GMT
View Forum Message <> Reply to Message

the function TreeCtrl::Find(Value) return a valid id after I remove the node.

I found the memory has managed by a Array<Item>. the remove function just push the node to the freelist by the field freelink.

So I think m.key=Null should be call in the remove function as following.

```
void   TreeCtrl::RemoveSubtree(int id)
{
   Item& m = item[id];
   if(m.sel)
      selectcount--;
   if(m.linei == cursor)
      cursor = item[m.parent].linei;
   if(m.ctrl)
      m.ctrl->Remove();
   m.key=Null
   m.value = Null;
   m.image = Null;
   RemoveChildren(id);
   m.freelink = freelist;
   freelist = id;
   m.free = true;
}
```

Subject: Re: A [TreeCtrl] bug
Posted by mirek on Mon, 03 Mar 2008 18:17:55 GMT
View Forum Message <> Reply to Message

Thanks for the bug report!

Well, that is not bad, but IMO

```
int    TreeCtrl::Find(Value key)
{
 for(int i = 0; i < item.GetCount(); i++)
  if(!item[i].free && Get(i) == key)
   return i;
 return -1;
}
```

is correct fix (well, Null key is unusual, but not impossible).

Note that Null asignments there are rather intended to release the memory....

Please check whether things work here (by replacing Find and commenting key = Null)!

Mirek

Subject: Re: A [TreeCtrl] bug
Posted by huanghuan on Tue, 04 Mar 2008 10:52:55 GMT
View Forum Message <> Reply to Message

Thank you. I will check it in next version.
if key = Null is only an option and not added. the following may half time complex in invalid key situation.

```
int    TreeCtrl::Find(Value key)
{
 for(int i = 0; i < item.GetCount(); i++)
  if(item[i].free)
  {
   if(Get(i) == key)
    return -1;
  }
  else if(Get(i) == key)
   return i;
 return -1;
}
```

this is not good. key = Null is nessary. for memory release.
I hope add the "key = Null".

Subject: Re: A [TreeCtrl] bug
Posted by mr_ped on Tue, 04 Mar 2008 11:38:38 GMT
View Forum Message <> Reply to Message

I didn't check TreeCtrl source nor do I know how it works, but if some memory spot is marked as "free", the value can be there undefined?
So if you find there the key you are searching for, it does not exclude the case of that particular key to be also found on some other spot, where the item[i].free == false?

In such case your optimization will break the result.

And it contains duplicate code... I would refactor it anyway to:

```
int    TreeCtrl::Find(Value key)
{
 for(int i = 0; i < item.GetCount(); i++)
  if (Get(i) == key)
   return item[i].free ? -1 : i;
 return -1;
}
```
Still I think it may be broken in the case of some valid key being also stored second time in some "free" item. (But I'm too lazy to check TreeCtrl source and learn how it works and how it is used.)

Quote:key = Null is nessary. for memory release.
Why? I don't see any connection between those two in C++.
Although I admit it's a good practice in C++ to always null your pointers after you released the allocated memory they are pointing to, so you don't access the freed memory by accident and rather get exception from null pointer, but actually it's "memory release for key = null", not "key = null for memory release", the reason and the consequence are the other way around.

Subject: Re: A [TreeCtrl] bug
Posted by mrjt on Tue, 04 Mar 2008 12:01:47 GMT
View Forum Message <> Reply to Message

Sorry, double posted. See below.

Subject: Re: A [TreeCtrl] bug
Posted by mrjt on Tue, 04 Mar 2008 12:16:28 GMT
View Forum Message <> Reply to Message

As far as I can see Mirek's version is the correct fix here.
luzr wrote on Mon, 03 March 2008 18:17Note that Null asignments there are rather intended to release the memory....
mr_ped wrote on Tue, 04 March 2008 11:38Quote:key = Null is nessary. for memory release.
Why? I don't see any connection between those two in C++.
Although I admit it's a good practice in C++ to always null your pointers after you released the allocated memory they are pointing to, so you don't access the freed memory by accident and rather get exception from null pointer, but actually it's "memory release for key = null", not "key = null for memory release", the reason and the consequence are the other way around.
I believe this is because both key and value are Value types that may contain reference-counted objects (such as Image), and setting Null is required to decrement this count and (perhaps) free the memory. So setting Null is can be the cause of memory release.

But if this is the case then surely 'key = Null' should be added? I admit it's unlikely that a reference counted object would be used as a Key, but it seems to be cost-free and doesn't break any code so why not do it?

I could be wrong about all this though, my understanding of this aspect of Upp is somewhat

limited.

James

Subject: Re: A [TreeCtrl] bug
Posted by mirek on Tue, 04 Mar 2008 14:20:20 GMT
View Forum Message <> Reply to Message

mrjt wrote on Tue, 04 March 2008 07:16
I believe this is because both key and value are Value types that may contain reference-counted objects (such as Image), and setting Null is required to decrement this count and (perhaps) free the memory. So setting Null is can be the cause of memory release.

Right on the spot... This is only minor optimization, but we want to consume as little memory as possible, right?

Quote:
But if this is the case then surely 'key = Null' should be added?

Definitely! It was added too.

Quote:
I admit it's unlikely that a reference counted object would be used as a Key

It is Value, so it is always reference counted... (I am considering to introduce "small value optimization", but numbers do not seem to match....)

Quote:
I could be wrong about all this though, my understanding of this aspect of Upp is somewhat limited.

Your understanding is near to perfect, as usual

Mirek
