Subject: broken pick semantics
Posted by crydev on Wed, 14 Nov 2012 22:23:25 GMT
View Forum Message <> Reply to Message

I'm building a movie managing program. I have a settings dialog that loads data from a global variable which is an instance of a class that manages an XML file. As long as I keep the loading of data inside the constructor of the dialog: there is no problem. When I create a new function that loads the data, for example, to reload the data every time I reopen the dialog I get a "broken pick semantics" error. My code is as following:

```
ConfigurationStorage MovieManagerConfiguration; // global variable that holds the settings data

void SettingsDialog::Reload()
{
 mDirectoryList.Clear();
 auto x = MovieManagerConfiguration.SyncDirectories.GetCount(); // this throws the semantics error!
 for (int i = 0; i < x; i++)
 {
  mDirectoryList.Add(MovieManagerConfiguration.SyncDirectories[i]);
 }
}
```

The Vector<String> SyncDirectories contains 1 string variable at that moment. If I put the loop code back inside the dialog constructor the error does not appear. I call the dialog as following:

Inside the constructor of my main window:

```
mSettingsDialog.WhenSettingsChanged = THISBACK(SettingsChanged);
```

Inside the open button:

```
void MovieManager::Options()
{
 mSettingsDialog.Reload();
 mSettingsDialog.Execute();
}
```

Why am I getting this error? How can I fix it? Am I maybe doing my settings management the wrong way by using a global class instance like this? I am busy on C++ while always have been doing C# so I will believe it if I am not doing the correct thing here.

Subject: Re: broken pick semantics
Posted by dolik.rce on Thu, 15 Nov 2012 03:16:48 GMT
View Forum Message <> Reply to Message

Hi crydev

Your problem with broken pick semantics is not actually on the line with GetCount(). That is just coincidentally the first place when you try to access picked container. You will have to look in the code that is executed before. Check for any copy/assignment of SyncDirectories to another Vector. Vector is by default using pick semantics, so when you copy it, the original content is destroyed.

Read the NTL tutorial, (especially section 3) for basic explanation of picking and also Transfer semantics and Pick behavior explanation for further details. It is some heavy reading, but necessary to understand the U++ containers. When I started to work with U++, I had to read it at least once a day for a week, before I got used to it and understood why is it so great

Also if you post more of the related code, I should be able to point you more exactly to where the problem started. It is definitely not in the parts that you posted.

Best regards,
Honza

Subject: Re: broken pick semantics
Posted by crydev on Thu, 15 Nov 2012 21:59:53 GMT
View Forum Message <> Reply to Message

I have read those topics and I understand how and why you did that. Though the information from those topics still couldn't help me find out my problem. I fixed it though! I created a reference to the global variable, and the error did not pop up anymore.

The problem didn't reside in the GetCount() in the above post as you said, the problem resided in my synchroniser:

```
void MovieSynchroniser::Synchronise(Vector<String> pDirectories)
{
 for (int i = 0; i < pDirectories.GetCount(); i++)
 {
  GetFiles(pDirectories[i]);
 }
}
```

This function was called using the global variable as parameter. I changed the function definition into:

void MovieSynchroniser::Synchronise(Vector<String>& pDirectories)

It makes sense to me though. Because I used a normal value object as parameter it already copied the contents of the vector, which using pick semantics, resulted in the original vector being empty. By using a reference pointer to the original vector, the pick assignment is not used, avoiding this problem. Am I correct?

Subject: Re: broken pick semantics
Posted by dolik.rce on Fri, 16 Nov 2012 06:31:54 GMT
View Forum Message <> Reply to Message

crydev wrote on Thu, 15 November 2012 22:59It makes sense to me though. Because I used a normal value object as parameter it already copied the contents of the vector, which using pick semantics, resulted in the original vector being empty. By using a reference pointer to the original vector, the pick assignment is not used, avoiding this problem. Am I correct?Yes, when you just pass a reference around, it is still one object and no copying is necessary. It is actually (IMHO) better practice, to avoid copies by using references everywhere where possible.

Honza
