Subject: (Possible) Serialization issue
Posted by Mindtraveller on Wed, 30 Jul 2008 05:14:21 GMT
View Forum Message <> Reply to Message

Short description: while loading binary data from file, stream.IsStoring() returns true and stream.IsLoading() returns false.

```
Compatible testcase:class TestClass
{
public:
 void Serialize (Stream &stream);

private:
 int b;
};

void TestClass::Serialize(Stream &stream)
{
 if (stream.IsStoring())
 {
  int a = 0; //BREAKPOINT STOP HERE!
 }
 if (!stream.IsLoading())
 {
  int a = 1; //BREAKPOINT STOP HERE!
 }

 stream % b;
}

CONSOLE_APP_MAIN
{
 TestClass testClass;
 LoadFromFile(testClass, ".config");
}
```

Recommended behaviour: in a number of situations it is vital to know if one loads or saves data. Correct IsStoring/IsLoading behaviour here is welcome.

Subject: Re: (Possible) Serialization issue
Posted by mirek on Wed, 30 Jul 2008 08:23:47 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Wed, 30 July 2008 01:14Short description: while loading binary data from file, stream.IsStoring() returns true and stream.IsLoading() returns false.

Compatible testcase:class TestClass

```
{
public:
 void Serialize (Stream &stream);

private:
 int b;
};

void TestClass::Serialize(Stream &stream)
{
 if (stream.IsStoring())
 {
  int a = 0; //BREAKPOINT STOP HERE!
 }
 if (!stream.IsLoading())
 {
  int a = 1; //BREAKPOINT STOP HERE!
 }

 stream % b;
}

CONSOLE_APP_MAIN
{
 TestClass testClass;
 LoadFromFile(testClass, ".config");
}
```

Recommended behaviour: in a number of situations it is vital to know if one loads or saves data. Correct IsStoring/IsLoading behaviour here is welcome.

It is a little bit more complicated.

This is "load failure" protection. LoadFromFile first *stores* data in temporary String buffer, then attempts to load them from the file. If load fails, it restores the backup copy.

```
bool Load(Callback1<Stream&> serialize, Stream& stream) {
 StringStream backup;
 backup.SetStoring(); // <<<< this is it :)
 serialize(backup);
 ASSERT(!backup.IsError());
 stream.SetLoading();
 stream.LoadThrowing();
 try {
  CheckedSerialize(serialize, stream);
 }
 catch(LoadingError) {
```

```
  backup.Seek(0);
  backup.SetLoading();
  serialize(backup);
  ASSERT(!backup.IsError());
  return false;
 }
 return true;
}
```

Note that if this is not what you want, do not use LoadFromFile

Mirek

Subject: Re: (Possible) Serialization issue
Posted by Mindtraveller on Fri, 01 Aug 2008 21:10:50 GMT
View Forum Message <> Reply to Message

It`s kind of big surprising functionality for a function with such a name. Then I propose renaming it into something like LoadFromFileWithBackup. You know, the difference between function name anf functionality should be minimized. Each library is a black box. Less surprises - more user satisfied.
Besides, I propose adding "simple" LoadFromFile alternative. This would be convenient and effective way of loading configuration. Without surprises.

More of that. Sometimes (while upgrading application and adding new objects to serialization queue end) it is OK to read all the data possible. Instead of rejecting all the file and loosing all the old choices made. In this case LoadFromFile could read everything it can and it is right IMO.

Subject: Re: (Possible) Serialization issue
Posted by mirek on Sun, 03 Aug 2008 12:11:42 GMT
View Forum Message <> Reply to Message

[quote title=Mindtraveller wrote on Fri, 01 August 2008 17:10]
Besides, I propose adding "simple" LoadFromFile alternative.
[quote]

Well, that is just one-liner:

object.Serialize(FileIn(filename));

Mirek

Subject: Re: (Possible) Serialization issue
Posted by Mindtraveller on Sun, 03 Aug 2008 20:03:41 GMT
View Forum Message <> Reply to Message

Thanx. And what about renaming? I understand compitibility issues, but don`t you consider "no surprises" idea importance?

Subject: Re: (Possible) Serialization issue
Posted by mirek on Sun, 03 Aug 2008 21:21:26 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Sun, 03 August 2008 16:03Thanx. And what about renaming? I understand compitibility issues, but don`t you consider "no surprises" idea importance?

Well, I do not see any surprise here. LoadFromFile is a complex function supposed to manage loading from files, using Serialize functions/methods.

All Serialize functions/methods must be 2-way anyway. Creating backup does not cause any state change in objects (at least, as long as Serialize are correct). And when error is encountered, it just "rollbacks" all changes.

Besides, renaming would break a lot of code.

Mirek

Subject: Re: (Possible) Serialization issue
Posted by bytefield on Mon, 04 Aug 2008 07:36:26 GMT
View Forum Message <> Reply to Message

Well, what if we use a "dirty" solution and instead of renaming it we declare a pointer to Load function and then declare Load as deprecated and after a while replace that pointer with content from Load, resulting a new function LoadFromFileWithBackup?

```
// Solution
bool (*LoadFromFileWithBackup)(Callback1<Stream&> serialize, Stream& stream) = & Load;
```

and after a while...

```
bool LoadFromFileWithBackup(Callback1<Stream&> serialize, Stream& stream)
{
 StringStream backup;
 backup.SetStoring();
 serialize(backup);
 ASSERT(!backup.IsError());
 stream.SetLoading();
```

```
 stream.LoadThrowing();
 try {
  CheckedSerialize(serialize, stream);
 }
 catch(LoadingError) {
  backup.Seek(0);
  backup.SetLoading();
  serialize(backup);
  ASSERT(!backup.IsError());
  return false;
 }
 return true;
}
```

Subject: Re: (Possible) Serialization issue
Posted by cbpporter on Mon, 04 Aug 2008 08:10:58 GMT
View Forum Message <> Reply to Message

I vote for leaving it as it is, because I pretty much depend on this feature. I never took the time to look into it and see why it recovers so well from wrong data (now I know), but I've come to rely on it and never noticed any performance loss.

Subject: Re: (Possible) Serialization issue
Posted by nixnixnix on Wed, 13 Aug 2008 19:11:28 GMT
View Forum Message <> Reply to Message

I retract my former comments

Nick

p.s. this is my attempt to setup a framework in which I can see the size of objects before I read them or alternatively just read past them.

```
int OCOE::GetSize()
{
 SizeStream s;

 Serialize1(s);

 int bytes = s.GetSize();

 return bytes;
}
```

```
void OCOE::Serialize(Stream& s)
{
 int n;

 if(s.IsStoring())
 {
  n = GetSize(); // i.e. this is a variable sized object
 }

 s % n;

 Serialize1(s);
}

void OCOE::Serialize1(Stream& s)
{
 int version = VERSION_MAJOR*100000000 + VERSION_MINOR*1000000 /*+
VERSION_VAR*10000*/ + VERSION_COMMIT;
 int n = version;
 s % n; // so you can check it and do logic on it

 String sType = "OCOE";
 s % sType;

 s % m_fRoadBase;
 s % m_fRoadExtant;
 s % m_fRoadMaxGrad;
 s % m_fRoadMinRadius;
 s % m_fRoadX;
...
```

I don't think the SizeStream::GetSize() function does what I hoped it does though so I need to keep looking for an easily maintainable solution...

Subject: Re: (Possible) Serialization issue
Posted by nixnixnix on Wed, 13 Aug 2008 19:32:27 GMT
View Forum Message <> Reply to Message

Is there anything that we can use that would act as a s.IsStoringToDisk() please?

Nick

p.s. the backup is a nice feature! never noticed that before

Subject: Re: (Possible) Serialization issue
Posted by mirek on Wed, 13 Aug 2008 20:24:32 GMT
View Forum Message <> Reply to Message

nixnixnix wrote on Wed, 13 August 2008 15:32Is there anything that we can use that would act as a s.IsStoringToDisk() please?

Ehm, what exactly do you mean with that? E.g. that the Stream is FileStream?

(In that case: s.IsStoring() && dynamic_cast<FileStream *>(&s))

Mirek

Subject: Re: (Possible) Serialization issue
Posted by mirek on Wed, 13 Aug 2008 20:28:12 GMT
View Forum Message <> Reply to Message

nixnixnix wrote on Wed, 13 August 2008 15:11
I don't think the SizeStream::GetSize() function does what I hoped it does though so I need to keep looking for an easily maintainable solution...

IMO, it should. It should count all 'Put's.

Mirek

Subject: Re: (Possible) Serialization issue
Posted by nixnixnix on Wed, 13 Aug 2008 22:49:40 GMT
View Forum Message <> Reply to Message

Yup it works and is the only way I can think of to mix open source and proprietary modules whilst maintaining document compatibility between open source and proprietary executables.

I probably wont patent it though so feel free to use it

For some reason I had to use

if(s.IsStoring() && dynamic_cast<FileStream *>(&s)!=NULL)

but it all works fine now. I know it doesn't really matter whether am storing or not but I just wanted to add the check for the sake of efficiency.

Thanks,

Nick

Subject: Re: (Possible) Serialization issue
Posted by Mindtraveller on Sat, 01 Nov 2008 10:53:38 GMT
View Forum Message <> Reply to Message

luzr wrote on Mon, 04 August 2008 01:21Mindtraveller wrote on Sun, 03 August 2008 16:03Thanx. And what about renaming? I understand compitibility issues, but don`t you consider "no surprises" idea importance?Well, I do not see any surprise here. Well, I returned to old topic because could finally summarize what is really a surprise here. The main surprise is when Serialize() is called and you can`t determine if it is REALLY a loading process or a storing. Why is it a problem? Loading (and saving) process may be followed by lengthy GUI updates or some other initializations including file/database work.

For example, on saving, my app generates a number of directories and files, then archives them with external utility. This was thought to be called rarely because it takes moderate amount of time and system resources (and these files have to be kept along a number of app runs). But each time I start application these files are re-created and re-archived because LoadFromFile() calls Serialize() with stream.IsStoring() == true. This situation will be complete surprise for those who use LoadFromFile() and didn`t read this topic.

IMO the idea of U++ error correction on serialize is wonderful. But user usually needs to know a little more about current serialization stage.
OK, I consider strong influence of compatibility requirements, so renaming LoadFromFile() is no more an option. The solution here could be adding some additional function like stream.IsXXXXXXX() which will guide real internal serializarion direction to be used internally by U++, while IsStoring()/IsLoading() could return general "user-level" serialization direction (IsLoading() always == true on LoadFromFile()).

Subject: Re: (Possible) Serialization issue
Posted by mirek on Sat, 01 Nov 2008 14:11:20 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Sat, 01 November 2008 06:53luzr wrote on Mon, 04 August 2008 01:21Mindtraveller wrote on Sun, 03 August 2008 16:03Thanx. And what about renaming? I understand compitibility issues, but don`t you consider "no surprises" idea importance?Well, I do not see any surprise here. Well, I returned to old topic because could finally summarize what is really a surprise here. The main surprise is when Serialize() is called and you can`t determine if it is REALLY a loading process or a storing.

How so? IsLoading/IsStoring says exactly that. Is IsLoading is true, you should load from stream, store to it otherwise.

Quote:
Why is it a problem? Loading (and saving) process may be followed by lengthy GUI updates or some other initializations including file/database work.

Indeed (note however that is mostly the case for Loading).

Quote:
For example, on saving, my app generates a number of directories and files, then archives them with external utility. This was thought to be called rarely because it takes moderate amount of time and system resources (and these files have to be kept along a number of app runs). But each time I start application these files are re-created and re-archived because LoadFromFile() calls Serialize() with stream.IsStoring() == true. This situation will be complete surprise for those who use LoadFromFile() and didn`t read this topic.

OK, I understand the problem. However, Serialize is indended for storing into stream or retrieving. Nothing about archiving with external utility as part of Serialize body was ever considered.... This is the same as e.g. managing RS232 port in Paint routine...

Quote:
The solution here could be adding some additional function like stream.IsXXXXXXX() which will guide real internal serializarion direction to be used internally by U++, while IsStoring()/IsLoading() could return general "user-level" serialization direction (IsLoading() always == true on LoadFromFile()).

But there really is no such thing. Real internal serialization direction is IsStoring for the backup pass of LoadFromFile. And it really does STORE data into stream.

Frankly, if you want things like this, why do not you just call Serialize directly? Well, it is 2 lines instead of one, but really quite simple...

```
FileOut out(myfile);
x.Serialize(out);
```

Mirek

Subject: Re: (Possible) Serialization issue
Posted by Mindtraveller on Mon, 03 Nov 2008 13:54:29 GMT
View Forum Message <> Reply to Message

I tried as you recommended loading 2-liner, but it didn`t work. I mean LoadFromFile works properly, but .Serialize on the same file gives me garbage.
My testcase:

```
VectorMap<String,String> config;
//site.Serialize(Stream &stream) { stream % config % ...; }
```

```
LoadFromFile(site, filenameConfig); //config here is OK

StoreToFile(site, filenameConfig); //testing with save and load again
FileIn f(filenameConfig);
site.Serialize(f); //config here is corrupted
```

Subject: Re: (Possible) Serialization issue
Posted by mrjt on Mon, 03 Nov 2008 14:09:17 GMT
View Forum Message <> Reply to Message

StoreToFile uses CheckedSerialize, which writes extra data to the file for error-detection.
Unfortunately this function doesn't have a public header, so you have three options:
1- Don't use StoreToFile (easiest)
2- Reproduce CheckedSerialize:

```
void CheckedSerialize(const Callback1<Stream&> serialize, Stream& stream)
{
 int pos = (int)stream.GetPos();
 stream.Magic(0x61746164);
 serialize(stream);
 stream.Magic(0x00646e65);
 pos = int(stream.GetPos() - pos);
 stream.Magic(pos);
}
```

3- Persuade Mirek that it should be public

Subject: Re: (Possible) Serialization issue
Posted by Mindtraveller on Tue, 04 Nov 2008 07:30:55 GMT
View Forum Message <> Reply to Message

mrjt wrote on Mon, 03 November 2008 17:09
1- Don't use StoreToFile (easiest) I continued trying to avoid LoadFromFile() with such a testcase:

```
String filenameConfig = "../../"+root+"/config";
 LoadFromFile(site, filenameConfig); // config is OK

 FileOut fOut;
 if (fOut.Open(filenameConfig)) // config is OK
 {
  site.Serialize(fOut);
  fOut.Close();
 }

 FileIn fIn;
 if (fIn.Open(filenameConfig))
 {
  site.Serialize(fIn);  // config corrupted!
```

```
 fIn.Close();
}
```

Even without using StoreToFile() I can`t use simple Serialize(), and U++ forces user to use StoreToFile/LoadFromFile. And it looks like Serialize(FileOut) uses the same magic header as Serialize(FileIn) doesn`t.

Subject: Re: (Possible) Serialization issue
Posted by mrjt on Tue, 04 Nov 2008 10:29:18 GMT
View Forum Message <> Reply to Message

MindtravellerEven without using StoreToFile() I can`t use simple Serialize(), and U++ forces user to use StoreToFile/LoadFromFile. And it looks like Serialize(FileOut) uses the same magic header as Serialize(FileIn) doesn`t.
I don't see how you're forced to use StoreToFile/LoadFromFile, I actually rarely do. And I can assure you that Serialize does work and that there are no magic headers.

I appreciate your frustration, but I'm pretty sure that the problem here is something in your code.
- In your test above, there are no checks on 'fOut.Open(filenameConfig)' failing. This would cause fin to read garbage.
- Check your serialization. If StoreToFile/LoadFromFile works then this should be fine but a simple test would be:

```
StringStream s;
int64 inpos, outpos;

// Write
site.Serialize(s);
outpos = s.GetPos();
//Read
s.SetLoading();
s.Seek(0);
site.Serialize(s);
inpos = s.GetPos();
ASSERT(inpos == outpos);
```

I hope that helps.
James

Subject: Re: (Possible) Serialization issue
Posted by Mindtraveller on Thu, 06 Nov 2008 12:33:42 GMT
View Forum Message <> Reply to Message

Thanks for the reply, I`ll check my code.
