Subject: Optional serialization techniques
Posted by Mindtraveller on Thu, 24 Feb 2011 15:27:49 GMT
View Forum Message <> Reply to Message

I have a version 1.0 of my application which serializes a VectorMap of some object into the file with StoreToFile. We of course know that if VectorMap object is being changed, the whole de-serialization is failed.
So here is my problem: I develop 1.1 version with objects which are slightly different. And actually what I want is that 1.1 version reads everything from config file ignoring the fact that objects can't be de-serialized completely (I just add more members since 1.0).
I don't want to make object members 'dynamic' (using VectorMap<String,Value> instead of plain members).
Yes, and I really don't want to use XML as speed is the most important in this case. And there is no problem just adding new members since new version (not removing old or replacing them).

Can you please suggest the most effective way of doing it?
Effective means the most quickly working while not rewriting all of U++ serialization code

Subject: Re: Optional serialization techniques
Posted by mirek on Fri, 25 Feb 2011 08:46:26 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Thu, 24 February 2011 10:27I have a version 1.0 of my application which serializes a VectorMap of some object into the file with StoreToFile. We of course know that if VectorMap object is being changed, the whole de-serialization is failed.
So here is my problem: I develop 1.1 version with objects which are slightly different. And actually what I want is that 1.1 version reads everything from config file ignoring the fact that objects can't be de-serialized completely (I just add more members since 1.0).
I don't want to make object members 'dynamic' (using VectorMap<String,Value> instead of plain members).
Yes, and I really don't want to use XML as speed is the most important in this case. And there is no problem just adding new members since new version (not removing old or replacing them).

Can you please suggest the most effective way of doing it?
Effective means the most quickly working while not rewriting all of U++ serialization code

I usually do:

```
void Foo::Serialize(Stream& s)
{
  int version = 0;
  s / version;
  s % x % y;
}
```

... and later I add 'z' to Foo:

```
void Foo::Serialize(Stream& s)
{
  int version = 1;
  s / version;
  s % x % y;
  if(version >= 1) {
    s % z;
  }
}
```

That said, it does not solve the problem all the time and generally, I would NOT recommend using binary serialization for permanent storage of important files. It is fine for configs (where if you loose one, it is not that bad) or for transfering data (e.g. over network).

Do not use it for documents

Mirek

Subject: Re: Optional serialization techniques
Posted by koldo on Fri, 25 Feb 2011 09:14:35 GMT
View Forum Message <> Reply to Message

I prefer XML serialization that is perfect for development and debugging. For end users that do not have to see the data, I just encrypt the XML file.

Subject: Re: Optional serialization techniques
Posted by chickenk on Fri, 25 Feb 2011 10:16:46 GMT
View Forum Message <> Reply to Message

Maybe what you search for is some kind of migration procedure so that your 1.0 serialized objects could be automatically migrated to 1.1 format before loading.

The migration layer is still to be done, but that would mean no code change in your config loader, just make it deserialize 1.1 objects. the additional code would be isolated.

If you want to keep your original 1.0 objects, you can save the 1.1 migrated file in the same place with a filename change so that both are available then. And when you detect a 1.0 version, you try to find or generate the 1.1 version.

that would allow an automatic upgrade of saved files for your users. Then, the migration code would simply include the default values for a specific migration.

A simple migration class would only add the required changes between 2 specific version. Then you can chain the migrations if several versions appeared.

For example, a user has 1.0 format and upgrades it software where the 1.2 version of the data format is used. Before loading, the 1.0 version is detected, and then:

1.0 --> migrate_100_to_110 --> 1.1 --> migrate_110_to_120 --> 1.2

This migration system is used by some existing software such as Ruby on Rails (which gave me this idea for you).

Not sure it can apply, but that could be something to dig into, and maybe a small bazaar package in sight?

Subject: Re: Optional serialization techniques
Posted by Mindtraveller on Fri, 25 Feb 2011 14:34:03 GMT
View Forum Message <> Reply to Message

Thanks everyone for the answers. I guess the problem is solved.

mirek wrote on Fri, 25 February 2011 11:46That said, it does not solve the problem all the time and generally, I would NOT recommend using binary serialization for permanent storage of important files. It is fine for configs (where if you loose one, it is not that bad) or for transfering data (e.g. over network).
Do not use it for documents
BTW, why? Looks like not a bad storage solution.

Subject: Re: Optional serialization techniques
Posted by mirek on Fri, 25 Feb 2011 18:42:24 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Fri, 25 February 2011 09:34Thanks everyone for the answers. I guess the problem is solved.

mirek wrote on Fri, 25 February 2011 11:46That said, it does not solve the problem all the time and generally, I would NOT recommend using binary serialization for permanent storage of important files. It is fine for configs (where if you loose one, it is not that bad) or for transfering data (e.g. over network).
Do not use it for documents
BTW, why? Looks like not a bad storage solution.

I guess the main problem is that it is so simple to serialize things that it is too easy to accidentally

break the format...

And, sometimes, the problem also is that it is impossible or too diffuclt to keep backward compatibility. Simple example would be forgetting to put 'version' somewhere...

Mirek

Subject: Re: Optional serialization techniques
Posted by tojocky on Sat, 26 Feb 2011 06:49:46 GMT
View Forum Message <> Reply to Message

mirek wrote on Fri, 25 February 2011 20:42

I guess the main problem is that it is so simple to serialize things that it is too easy to accidentally break the format...

And, sometimes, the problem also is that it is impossible or too diffuclt to keep backward compatibility. Simple example would be forgetting to put 'version' somewhere...

Mirek

Mirek, What about to migrate to a XML serialization? The things will be easy!
Your realization is very nice, but debugging and keeping backward compatibility is a little hard.

Subject: Re: Optional serialization techniques
Posted by mirek on Fri, 04 Mar 2011 09:01:50 GMT
View Forum Message <> Reply to Message

tojocky wrote on Sat, 26 February 2011 01:49mirek wrote on Fri, 25 February 2011 20:42

I guess the main problem is that it is so simple to serialize things that it is too easy to accidentally break the format...

And, sometimes, the problem also is that it is impossible or too diffuclt to keep backward compatibility. Simple example would be forgetting to put 'version' somewhere...

Mirek

Mirek, What about to migrate to a XML serialization? The things will be easy!

Uh, why? For what it is used, binary serialization is fine.

Quote:
Your realization is very nice, but debugging and keeping backward compatibility is a little hard.

I guess all this says is that you should not use it where backward compatibility is important.

Mirek
