

Forum: [Hack Forums](#)

Thread: [Converting AeonHack's Serializer to Java](#)

Systers - January 30, 2020, 4:05 pm

Could someone please help convert aeon's serializer into java? I tried it myself but it just ended up not working no matter how many hours of work I put into trying to fix it.

VB.NET: <https://github.com/aeonhack/archive/blob/master/sources/pastebin/serializer.vb>

C#: <https://gist.github.com/Virility/114369b1a1c654cb678d>

GenericSkid - January 30, 2020, 9:53 pm

Why would you want this when there are so many existing serialization approaches already written in Java?

Depending on what you're doing, somebody can recommend you a native java solution rather than this.

Systers - January 30, 2020, 10:17 pm

(Jan 30, 2020 10:53 PM)GenericSkid Wrote: [\[CITE\]](#) Why would you want this when there are so many existing serialization approaches already written in Java?

Depending on what you're doing, somebody can recommend you a native java solution rather than this.

I'm using it for AsynchronousSocketChannels and I want to use it because it would be extremely easy to use along with being small and efficient. If you have a native java serializer that works like this while being under 20kb that would be great.

MavenGradle - February 1, 2020, 12:00 am

(Jan 30, 2020 05:05 PM)Systers Wrote: [\[CITE\]](#) I tried it myself but it just ended up not working no matter how many hours of work I put into trying to fix it.

First of all, what exactly are you having difficulty with? Clearly some data types like uint, ushort etc. simply do not have a Java equivalent and as such you can simply use int, short etc. instead. While it is possible to work with unsigned values in Java, the declarations that are available in C# and VB.NET are not available here. More info here:

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

Second, I don't know why you'd want to incorporate this garbage into your code and bluntly copy it without understanding or knowing what it does. Immediately you can tell that this C#/VB.NET code is complete garbage and if AeonHack or whatever he's called now-a-days ever gets out of prison, I hope he realizes that. The reason why is because it's a pain to maintain, having to add switch cases for every new data type that he wants to support and it's not even following any of the OOP principles. There are so many things that could go wrong with his code, with his serializing and deserializing, it's incredible. Shit code.

Third, in Java, you can use an ObjectOutputStream to write objects to a ByteArrayOutputStream to achieve a similar result while supporting any Object type. As such, you don't need nearly as much code to achieve a similar result. Something like this:

```
Code    try (final ByteArrayOutputStream bos = new ByteArrayOutputStream()) {
```

```

    try (final ObjectOutputStream out = new ObjectOutputStream(bos)) {
        out.writeObject(object);
        return bos.toByteArray();
    }
}

```

Maybe you should learn more about how bytes operate and how serialization works in general, because I really don't understand how you could be having issues with something so trivial. This isn't even serialization in my opinion. Here's an example (total of 9.9kB) that I made just for you that is essentially a wrapper around the code above that can 'serialize' objects, depending on if they're available in this 'magical' list (because you wanted to stay true to the original), while adding their type and the length of the data to the 'packet':

ObjectSerializer:

TypeObject:

Example usage:

```

Code    int first = 10;
        int second = 20;
        String third = "Third";
        String fourth = "Fourth";

        byte[] data = serializer.serialize(first, second, third, fourth);
        List<TypeObject> result = serializer.deSerialize(data);

        for (int i = 0; i < result.size(); i++) {
            System.out.println(i + "\t:\t" + result.get(i).getObject());
        }

```

Here's an example byte package that contains a Java Integer with a value of 10:

Code

```

00000008000000051ACED0005737200116A6176612E6C616E672E496E746567657212E2A0A4F7
81873802000149000576616C7565787200106A6176612E6C616E672E4E756D62657286AC951D0
B94E08B02000078700000000A

```

If you dissect it, the first 4 bytes (00-00-00-08) are used to indicate the type, which is Integer. After that, 4 bytes are used to indicate the length of the actual object (00-00-00-51) which indicates that the actual data is 81 (decimal value of 00-00-00-51) bytes long. Finally, the actual byte data is presented to us and then the process is repeated for all additional objects that are in the total package. Obviously, this format will not be cross-compatible with other programming languages.

Noteworthy is that in a dynamic situation, you shouldn't even be using that Map<Integer, Object> to put artificial limitations in place, you should allow all objects to be 'serialized' (if you can even call it that, it's just a byte array of an object along with the length of said byte array in a single 'packet').

You can improve upon this design by decreasing the 4 bytes that are required to indicate the type and length. Preferably, you can make them dynamic in width, only requiring the bytes that are truly necessary. I'll leave that up to you though. It's really not hard to understand or create, since this took me like 30 minutes. You should also realize the limitations of my code above. If you don't, then I don't know what to tell you except that you should go back to school and learn and understand what it is that you're doing.

Additionally, you might want to look into adding checksums or a way to verify the byte array before

parsing it. Making the package size smaller can also be achieved by adding compression algorithms to the functionality. There are numerous ways how this can be done, but again, I'll leave that up to you. If you understand this code and what it does, it should be peanuts to turn this into something proper and fully dynamic. In fact, it took me longer to make this post than it took me to think of and write a complete implementation. [Now talk about reinventing the wheel](#) ?. Trust me, now-a-days most computers have at least 2MB to spare on their disk for a silly little library that you're too afraid to add. One more thing that's also noteworthy if you decide to go through with this would be to include a version indicator, so that if changes occur, a conversion mechanic (if applicable) can be called.

GenericSkid - February 3, 2020, 8:58 pm

Any reason why your impl has "typeList" as a Map? Looks like you forgot to refactor the name after changing the declared type.

MavenGradle - February 4, 2020, 10:43 am

(Feb 3, 2020 09:58 PM)GenericSkid Wrote: [\[CITE\]](#) Any reason why your impl has "typeList" as a Map ? Looks like you forgot to refactor the name after changing the declared type.

The name of that variable and methods that make use of the Map should be changed along with the name of the class itself (ObjectSerializer would be a better choice when names are concerned) but more importantly, the comments below the code are what matters and why using code similar to AeonHack makes no sense.

GenericSkid - February 4, 2020, 11:10 am

(Feb 4, 2020 11:43 AM)MavenGradle Wrote: [\[CITE\]](#) but more importantly, the comments below the code are what matters and why using code similar to AeonHack makes no sense. I'm well aware that the given Aeonhack code is less than ideal. I just skimmed the pastebins because I was curious what your implementation was. I knew you'd get around to this thread eventually ?

MavenGradle - February 4, 2020, 11:18 am

(Feb 4, 2020 12:10 PM)GenericSkid Wrote: [\[CITE\]](#) I'm well aware that the given Aeonhack code is less than ideal. I just skimmed the pastebins because I was curious what your implementation was. I knew you'd get around to this thread eventually ?

Corrected it for the OP regardless. It was a bit of a sleepless night/morning when I started writing the code. Nothing more than an exercise in stupidity and a good way to stay awake for an extra hour or two to write everything out ?.

AnonMouse - February 6, 2020, 6:36 am

it looks tricky but i am sure it can be done easily if i look at it later i will inbox you

corpsecorpsecorpse - April 23, 2020, 5:28 pm

This guy is a snitch ! Still in the game like a charme.... Come on you work with the Devil! Hahaha this guy is a legend.

About your problem this is not necessary for the javascript use the websocket better and more firendly maaaaan too heavy for a lot of config ! SmartPhone is a bitch.... Too expensive and nothing working. Stop this language hahaha loosing time so this is the real money braah.

youlded - has1 - April 24, 2020, 1:55 pm

Don't even bother trying to implement your own serialization format, it will be hard to maintain, slow, inefficient, and you will have to create an implementation for every language you want to use it with.

Use a standard, for example JSON:

Every language has a JSON binding, java, javascript, c#, vb.net, PHP, Go, swift, crystal, python,

ruby, etc.

PHP example:

<https://www.php.net/manual/de/function.json-encode.php>

C#

<https://docs.microsoft.com/de-de/dotnet/api/system.web.helpers.json.encode?view=aspnet-webpages-3.2>

Java:

<https://www.javatpoint.com/java-json-example>

as you can see, it's usually only 1 function call or a few lines.

If you want something even more "efficient" (that means, fewer bytes on the wire when sending the data, and also less time parsing the data) then I suggest you protobuf:

<https://developers.google.com/protocol-buffers>

Protocol buffers are created by Google to communicate between their own internal services.

I use protocol buffers myself in combination with gRPC for APIs: <https://grpc.io/>