Item 322
**git_comments:**

1. the aim to fix avro's bug https://issues.apache.org/jira/browse/AVRO-1891 bug address explain fix the avro logical type read and write

**git_commits:**

1. **summary:** Logical type use (#3900)
   **message:** Logical type use (#3900) ## Motivation To compromise avro‘s bug for avro filed logical type https://issues.apache.org/jira/browse/AVRO-1891 ## Modifications add some initialize class ## Verifying this change Add logical type test in AvroSchemaTest

**github_issues:**

**github_issues_comments:**

**github_pulls:**

1. **title:** Logical type use
   **body:** Motivation To compromise avro‘s bug for avro filed logical type https://issues.apache.org/jira/browse/AVRO-1891 Modifications add some initialize class Verifying this change Add logical type test in AvroSchemaTest Does this pull request potentially affect one of the following parts: If yes was chosen, please highlight the changes Dependencies (does it add or upgrade a dependency): (no) The public API: (no) The schema: (yes) The default values of configurations: (no) The wire protocol: (no) The rest endpoints: (no) The admin cli options: (no) Anything that affects deployment: (no)

**github_pulls_comments:**

1. run java8 tests
2. run Integration Tests
3. Merged in 2.3.1 at 0bc17d1

**github_pulls_reviews:**

1. Please define the dependency and version in top level Pom.xml in the dependenciesManagement section, to ensure we a have a single version
2. all right,I wiil do it
3. @merlimat @congbobo184 sorry, I PR #3856 add avro LogicalTypes.date()/timeMillis()/timestampMillis(), FieldSchemaBuilderImpl.java ``` // DATE, TIME, TIMESTAMP support from generic record case DATE: baseSchema = LogicalTypes.date().addToSchema(Schema.create(Schema.Type.INT)); break; case TIME: baseSchema = LogicalTypes.timeMillis().addToSchema(Schema.create(Schema.Type.INT)); break; case TIMESTAMP: baseSchema = LogicalTypes.timestampMillis().addToSchema(Schema.create(Schema.Type.LONG)); break; ``` No influence on each other ？
4. @ambition119 I think avro schema should use joda, #3856 should fix the pulsar schema. It shouldn't to change the avro schema.
5. just FieldSchemaBuilderImpl.java
6. We also need to update `distribution/server/src/assemble/LICENSE.bin.txt` with the new version for the dependency or the build will fail.
7. Additionally, I was suggesting to add the dependency here as part of `dependencyManagement` section. This is to force a single version of a particular artifact is used throughout the project
8. Once the dependency is set in `dependencyManagement`, we won't need to specify the version here.
9. > We also need to update distribution/server/src/assemble/LICENSE.bin.txt with the new version for the dependency or the build will fail. That's actually not a problem here :) because it's a test dependency so it's not included in binary distribution. As general rule though, we try to follow the `dependencyManagement` to pin the deps versions.

**jira_issues:**

1. **summary:** Generated Java code fails with union containing logical type
   **description:** Example schema: {code} { "type": "record", "name": "RecordV1", "namespace": "org.brasslock.event", "fields": [ { "name": "first", "type": ["null", {"type": "long", "logicalType":"timestamp-millis"}]} ] } {code} The avro compiler generates a field using the relevant joda class: {code} public org.joda.time.DateTime first {code} Running the following code to perform encoding: {code} final RecordV1 record = new RecordV1(DateTime.parse("2016-07-29T10:15:30.00Z")); final DatumWriter<RecordV1> datumWriter = new SpecificDatumWriter<>(record.getSchema()); final ByteArrayOutputStream stream = new ByteArrayOutputStream(8192); final BinaryEncoder encoder = EncoderFactory.get().directBinaryEncoder(stream, null); datumWriter.write(record, encoder); encoder.flush(); final byte[] bytes = stream.toByteArray(); {code} fails with the exception stacktrace: {code} org.apache.avro.AvroRuntimeException: Unknown datum type org.joda.time.DateTime: 2016-07-29T10:15:30.000Z at org.apache.avro.generic.GenericData.getSchemaName(GenericData.java:741) at org.apache.avro.specific.SpecificData.getSchemaName(SpecificData.java:293) at org.apache.avro.generic.GenericData.resolveUnion(GenericData.java:706) at org.apache.avro.generic.GenericDatumWriter.resolveUnion(GenericDatumWriter.java:192) at org.apache.avro.generic.GenericDatumWriter.writeWithoutConversion(GenericDatumWriter.java:110) at org.apache.avro.specific.SpecificDatumWriter.writeField(SpecificDatumWriter.java:87) at org.apache.avro.generic.GenericDatumWriter.writeRecord(GenericDatumWriter.java:143) at org.apache.avro.generic.GenericDatumWriter.writeWithoutConversion(GenericDatumWriter.java:105) at org.apache.avro.generic.GenericDatumWriter.write(GenericDatumWriter.java:73) at org.apache.avro.generic.GenericDatumWriter.write(GenericDatumWriter.java:60) at org.brasslock.avro.compiler.GeneratedRecordTest.shouldEncodeLogicalTypeInUnion(GeneratedRecordTest.java:82) at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62) at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43) at java.lang.reflect.Method.invoke(Method.java:498) at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50) at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12) at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47) at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17) at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325) at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78) at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57) at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290) at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71) at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288) at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58) at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268) at org.junit.runners.ParentRunner.run(ParentRunner.java:363) at org.junit.runner.JUnitCore.run(JUnitCore.java:137) at com.intellij.junit4.JUnit4IdeaTestRunner.startRunnerWithArgs(JUnit4IdeaTestRunner.java:117) at com.intellij.junit4.JUnit4IdeaTestRunner.startRunnerWithArgs(JUnit4IdeaTestRunner.java:42) at com.intellij.rt.execution.junit.JUnitStarter.prepareStreamsAndStart(JUnitStarter.java:253) at com.intellij.rt.execution.junit.JUnitStarter.main(JUnitStarter.java:84) at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62) at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43) at java.lang.reflect.Method.invoke(Method.java:498) at com.intellij.rt.execution.application.AppMain.main(AppMain.java:147) {code} The failure can be fixed by explicitly adding the relevant conversion(s) to DatumWriter / SpecificData: {code} final RecordV1 record = new RecordV1(DateTime.parse("2007-12-03T10:15:30.00Z")); final SpecificData specificData = new SpecificData(); specificData.addLogicalTypeConversion(new TimeConversions.TimestampConversion()); final DatumWriter<RecordV1> datumWriter = new SpecificDatumWriter<>(record.getSchema(), specificData); final ByteArrayOutputStream stream = new ByteArrayOutputStream(AvroUtil.DEFAULT_BUFFER_SIZE); final BinaryEncoder encoder = EncoderFactory.get().directBinaryEncoder(stream, null); datumWriter.write(record, encoder); encoder.flush(); final byte[] bytes = stream.toByteArray(); {code}

**jira_issues_comments:**

1. Had a quick look at this. The scope of this problem is bigger than I expected. It looks like that current {{SpecificCompiler}} and {{SpecificDatumWriter/Reader}} haven't considered the logical type conversion in some complex types (array, map, union) yet. Will try to find a way to fix this problem.
2. I'm not sure that is entirely correct. GenericData methods handle most of these cases; few changes were needed for specific assuming that the data model has the conversion. I think the problem is that specific adds [alternate lookups for conversions|https://github.com/apache/avro/blob/master/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificDatumReader.java#L109] that aren't currently called in methods like [{{resolveUnion}}|https://github.com/apache/avro/blob/master/lang/java/avro/src/main/java/org/apache/avro/generic/GenericData.java#L686]. That only uses the conversions from the data model. I think we just need to update a few more methods to know about the alternate lookup. Is that what you meant?
3. **body:** Hi [~rdblue], thanks for the reply! I agree that we need to modify some methods in SpecificDatumReader/Writer to use conversions contained in SepcificRecord. Problem is that currently the {{SpecificCompiler}} doesn't generate conversions for complex type fields: https://github.com/apache/avro/blob/master/lang/java/compiler/src/main/java/org/apache/avro/compiler/specific/SpecificCompiler.java#L647-L672 The complex types themselves don't contain a "logical type". Instead, it is their element types that can contain a logical type. We may need to modify the compiler to include these scenarios. I have a quick thought about this, ARRAY and MAP type are OK. They contain only one element type. But for UNION, it is possible that there are multiple element types in it. [The current {{Conversion}} array|https://github.com/apache/avro/blob/master/lang/java/compiler/src/main/java/org/apache/avro/compiler/specific/SpecificCompiler.java#L647-L672] in {{SpecificRecord}} cannot hold this. I was considering changing it as {{java.util.List<org.apache.avro.Conversion<?>>[] conversions}}, but looks like this is a non compatible change. Now I tend to add another field specially for union fields (need to change {{getConversion}} to determine which field to look at -- bad for performance}}. What do you think about this?
   **label:** code-design
4. **body:** Things can become even more complicated. AVRO supports multi-dimensional array, and any complex type can be included in another complex type. This makes it necessary for specific compiler to check all the levels of an idl definition and find out all the used conversions. Currently the conversions are stored on a per field basis. This may not be suitable for the scenarios mentioned above. Instead, I propose the store all the conversions (from any level) in a set so that application can use it when necessary.
   **label:** code-design
5. [~rdblue], [~busbey], could you please help confirm whether we need support embedded complex type in specific records? Such as: {code} union { array<string>, map<int> } {code}
6. yes, we should expect those to work.
7. **body:** Might it not be simpler to take advantage of the conversion code that's already in GenericDatumWriter (that pre-dates AVRO-1684)? We could do this by adding a constructor to SpecificDatumWriter that takes a SpecificRecord, so it uses the SpecificData instance from the generated class (which should have the conversions added to it). The downside is that users will need to use this constructor otherwise they'll get an exception if the generated class uses logical types. However, this is cleaner than trying to support a parallel set of conversions in SpecificRecordBase that SpecificDatumWriter has to know about.
   **label:** code-design
8. **body:** {quote} We could do this by adding a constructor to SpecificDatumWriter that takes a SpecificRecord, so it uses the SpecificData instance from the generated class (which should have the conversions added to it). {quote} # At this moment, only the generated {{SpecificRecord}} contains a {{SpecificData}} member, and this member is private. {{SpecificRecord}} itself doesn't provide any method that returns a related {{SpecificData}}. Adding such a method is not backward compatible. # We still need to consider the situation where a {{SpecificRecord}} is nested in another {{SpecificRecord}}, and these 2 records may use different set of conversions (for example one use decimal conversion while the other uses date conversion). {{SpecificDatumWriter/Reader}} need to process such a scenario. Currently I am thinking of moving the conversion management function away from {{Specific/GenericData}} and put it into another class (say ConversionResolver), and pass its objects as a parameter to Specific/Generic datum readers/writers. But I haven't got time to carefully examine this idea. Will find some time this week to do this.
   **label:** code-design
9. Re 1. We could add a static method {{getClassDataModel()}} to generated classes, so newly generated classes could use it to get the {{SpecificData}} object for the {{SpecificRecord}}. Re 2. We could have a fixed set of conversions, just like {{SpecificCompiler}} does. I've attached a patch to show what this might look like. I also had a look at going down the "alternative conversion lookup" route, but apart from the field position problem (which can be fixed by using maps), there's a problem with {{resolveUnion}}. When {{resolveUnion}} is called there is no {{SpecificRecordBase}} object to get the conversions from. I can't see a way around that, but perhaps others have ideas.
10. {quote} When resolveUnion is called there is no SpecificRecordBase object to get the conversions from. {quote} We can add another {{resolveUnion}} method that takes an extra parameter that holds all the conversions, and overwrites the {{resolveUnion}} method in {{SpecificDatumWriter}}. I will my patch (not completed yet) as well. Please help have a look. The idea is to overloads the {{writeRecord}} in {{SepecificDatumWriter}} and utilize the conversions used in the generated specific record for writing data. A similar thing is done in {{SepcificDatumReader}} as well. BTW, if we go with your patch, we will have to check the {{SpecificCompiler}} parameters to determine whether we should include {{DecimalConversion}} to the variable Model$.
11. Attach another patch that fixes the reading problems
12. I like the idea of encapsulating the logic for doing logical type conversions (ConversionResolver), although this has the effect of creating another class that doesn't really need to be public. The modifications to support union look fairly minor. Are any further changes needed for arrays and maps? It looks like RecordBuilderBase has an incompatible change to defaultValue(). We need a test for multiple nested logical types with generated code, as well as ones for arrays and maps of logical types. (BTW the v3 patch doesn't apply to latest master for me.) > BTW, if we go with your patch, we will have to check the SpecificCompiler parameters to determine whether we should include DecimalConversion to the variable Model$. The idea was that you'd always include all known conversions (the same set that SpecificCompiler uses). Would that not work?
13. **body:** I don't see the advantage of adding ConversionResolver. We'll be stuck with a bunch of deprecated methods for a number of releases or we break things incompatibly. Does it add any expressive power, or is it just done for cleanliness? In Tom's patch I wish there were less generated code. Couldn't a lot of that be done in the base classes? In general, we should try to minimize generated code whenever possible.
    **label:** code-design
14. {quote} although this has the effect of creating another class that doesn't really need to be public {quote} I will try to make this class non-public. {quote} The modifications to support union look fairly minor. Are any further changes needed for arrays and maps? {quote} We need some changes to support writing records and record fields. The patch has already covered this. There should be no further change needed for arrays and maps. {quote} It looks like RecordBuilderBase has an incompatible change to defaultValue(). {quote} I deleted the previous {{defaultValue}} method, thinking it is only needed in by the generated code and now the generated code doesn't use it any more. I forgot this was a protected method and there has been a release that exposes this change. Deleting it would bring a backward compatibility problem. I will add it back. Thanks for pointing this out. And sorry for uploading a broken patch. I will try to correct it in next version. {quote} We need a test for multiple nested logical types with generated code, as well as ones for arrays and maps of logical types. (BTW the v3 patch doesn't apply to latest master for me.) {quote} Yes, I am working on this. I am thinking of changing class {{TestRecordWithLogicalTypes}} to make it contain more types of nested types. {quote} The idea was that you'd always include all known conversions (the same set that SpecificCompiler uses) {quote} Users can pass a parameter to {{SpecificCompiler}} to control whether it uses {{BidDecimal}} or {{ByteBuffer}} for decimal types. That is, the {{DecimalConversion}} is not always used in generated code. (This is to keep backwards compatibility. We may not be able to assume {{DecimalConversion}} is always used.
15. {quote} I don't see the advantage of adding ConversionResolver. We'll be stuck with a bunch of deprecated methods for a number of releases or we break things incompatibly. Does it add any expressive power, or is it just done for cleanliness? {quote} The idea of my patch is to extract find out the conversions used in a child of {{SpecificRecordBase}} and merged them with those added to the {{Generic/SpecificData}} in datum reader/writer so that the record fields can be processed properly without users having to manually adding the conversions. I extracted {{ConversionResolver}} out just to avoid creating heavy {{Generic/SpecificData}} objects. [~tomwhite] mentioned that we could make {{ConversionResolver}} non-public. I am thinking about making it an inner class of GenericData. This way, we might be able to avoid exposing {{ConversionResolver}} and marking a bunch of methods deprecated. I will upload another patch later.
16. **body:** I have another look into the code, and find that we might have to extract {{ConversionResolver}}, because: # {{Generic/SpecificData}} are public classes, and they are not final. Applications may have extended from them, which makes it hard to create them (especially their children) on the fly, which is necessary if we want to add conversions when writing/reading children of {{SpecificRecordBase}}. On the other hand, {{ConversionResolver}} is a new class, we can define it as final, which make it easy to create objects. # creating {{Generic/SpecificData}} object in specific datum reader/writer is confusing, especially when the reader/writer already has a {{data}} member. Using {{ConversionResolver}} makes code much clearer. # Actually we don't have to mark those add/get conversion

methods in {{GenericData}} as deprecated. They are still useful and provide a convenient way for applications to manage conversions. What are you opinions on this, [~cutting] and [~tomwhite]?
**label:** code-design
17. Just found the {{defaultValue}} method I added has a bug. I guess you were actually mentioning to this bug when you said that my modification was incompatible. I will fix it. Thanks for pointing this out.
18. Upload a new patch that fixes the defaultValue bug and can be applied to master branch
19. **body:** I think all this requires is keeping a set of conversions that should be applied when reading or writing a specific class. Unlike generic where the conversions are determined by the data model at runtime, the conversions that should be used for a specific class are determined at compile time. We have the benefit of knowing that the compiler either added conversions for all instances of a logical type, or for none of them. So we only need to know the set of conversions the compiler had set up when a class was compiled. Rather than relying on the set of conversions the SpecificData instance has configured, I think we should keep the set of conversions for the class being written or read. So we don't need to change how SpecificData looks up conversions, just the way the SpecificDatumReader/Writer does to avoid looking them up in the data model. (I agree with Doug and don't see an advantage of adding a conversion resolver.) What about this: * Maintain a thread-local reference to the current specific record class in SpecificDatumReader * Add a static conversion map to each specific class with its conversions (generated code) * Add conversion lookup methods to GenericDatumReader that delegate to GenericData * Override the conversion lookup methods in SpecificDatumReader that use the current record class's set of conversions instead. This way, there are no changes to how the data model lookups work, little generated code (just an annotation to conversion map), and few changes to the datum reader and writers. What do you guys think? I think this would be a bit smaller patch. I'll try to put it together tomorrow if I have time.
**label:** code-design
20. [~rdblue], you idea looks good to me! {quote} Maintain a thread-local reference to the current specific record class in GenericDatumReader {quote} I am also thinking about using a thread local variable, because my patch makes data reader/writer non thread-safe. We may need to keep a stack of class references, which will be helpful when there are nested records. {quote} # Add conversion lookup methods to GenericDatumReader that delegate to GenericData # Override the conversion lookup methods in SpecificDatumReader that use the current record class's set of conversions instead. {quote} This is a great idea! Looking forward to your patch! (For backward compatibility, the conversion lookup methods in SpecificDataumReader/Writer may need to fallback to those in GenericDatumReader/Writer if the generated class doesn't have the static conversion map -- for legacy classes I mean)
21. +1 to determining the conversions for specific at compile time.
22. GitHub user rdblue opened a pull request: https://github.com/apache/avro/pull/118 AVRO-1891: Fix specific nested logical types This changes the datum readers to use `getConversionFor` and `getConversionByClass` instance methods that delegate to the data models. This allows specific to use a set of conversions determined by the current record class to pick up the conversions that were present when it was compiled. You can merge this pull request into a Git repository by running: $ git pull https://github.com/rdblue/avro AVRO-1891-fix-specific-nested-logical-types Alternatively you can review and apply these changes as the patch at: https://github.com/apache/avro/pull/118.patch To close this pull request, make a commit to your master/trunk branch with (at least) the following in the commit message: This closes #118 ---- commit 6c04cb3f3daec9e87e49d440faa35a248767fb3b Author: Ryan Blue <blue@apache.org> Date: 2016-09-04T21:54:40Z AVRO-1908: Fix TestSpecificCompiler reference to private method. AVRO-1884 changed makePath to a private method from a package-private static method. This broke the test that references the method in IPC. The fix is to make the instance method package-private and update the test to use an instance of SpecificCompiler. commit 31be7fa9d4589e7553d274d089aa3bf97c0297b3 Author: Ryan Blue <blue@apache.org> Date: 2016-09-05T19:15:09Z AVRO-1891: Add conversion getters to datum readers. This adds getConversionFor and getConversionByClass to GenericDatumReader so that SpecificDatumReader can override them to resolve conversions using the set of conversions used when a specific class was compiled, determined by the CONVERSIONS static variable or by adding the conversions for each field returned by getConversion(String). commit cf5bff3b6f7d312a65c28dbb053e6dbb6ab61c58 Author: Ryan Blue <blue@apache.org> Date: 2016-09-05T19:44:13Z AVRO-1891: Java: Add converions to compiled specific records. This adds a CONVERSIONS static field with the converisons that the specific compiler used to produce a record. Together with the last commit, this will enable new specific classes to use nested logical types. ----
23. **body:** I've posted a PR with the proposed changes. It still needs tests to validate it works for maps, arrays, and unions, but it demonstrates the idea.
**label:** test
24. The patch looks great, [~rdblue]! I add a few comments to it.
25. Thanks for posting a patch Ryan. How would it work for the write side?
26. Much the same way, by keeping track of the conversions for the record currently being written or delegating to the data model's types for generic and reflect. When I get time, I'll finish the write side and add tests (though if anyone else feels like continuing to get it out quicker, I'm fine with that).
27. Hi [~rdblue] and [~Yibing]. Is this issue the same as : https://stackoverflow.com/questions/45581437/how-to-specify-converter-for-default-value-in-avro-union-logical-type-fields ? Additionally, any plans to merge the PR? As it currently stands, it seems we can't use code generation with a union of null and a logical type, which is really annoying...
28. Can someone give me a synopsis of current status? do we believe this is implemented enough for review?
29. Agreed this is a very annoying issue when we cant have "null" for built in types like timestamp-millis and decimal.
30. **body:** There has been no activity on this ticket for 14 months.  Is this PR abandoned?  Is there another workaround for this bug?  It *is* very annoying that built in types are not fully supported.
**label:** code-design
31. wheelerlaw commented on issue #118: AVRO-1891: Fix specific nested logical types URL: https://github.com/apache/avro/pull/118#issuecomment-375532605 What happened to this? This seems pretty major, because without it, unions with built-in logical types are **broken**. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
32. rdblue commented on issue #118: AVRO-1891: Fix specific nested logical types URL: https://github.com/apache/avro/pull/118#issuecomment-375714156 @wheelerlaw, feel free to pick this up. I'll help review the changes. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
33. If union with logical type is not well supported, please *add a note to the Avro spec page*. We spend several days debug with client only to find it's a issue with Avro lib. Thanks!
34. Hi [~rdblue], Do we have an estimate on when this JIRA is going to be merged?
35. lewisdawson commented on issue #118: AVRO-1891: Fix specific nested logical types URL: https://github.com/apache/avro/pull/118#issuecomment-412320009 Is there still any intention of finishing & merging this PR? I ran into this today when trying to serialize avro data to a Kafka stream. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
36. scatrin opened a new pull request #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329 Fixes AVRO-1891, AVRO-2065 Adds support for pluggable conversion classes -- new feature Changes the compiler to add used conversions to the SpecificData instance already stored in the generated class. Adds static utility methods in SpecificData for getting the correct SpecificData instance from a class or a schema (using reflection). Adds calls to these utility methods from contexts where the schema is known. Also: Adds a possibility to add custom conversion classes to the compiler using a parameter to the Maven plugin. The custom conversion instances used are stored in the generated class with the same mechanism as above. They must of course be on the classpath at compile and serde time and whenever else the generated class is used. This feature is not required for the bug fixes to work but the addition is not large and IMO it adds nice value. Not implemented yet: * Custom conversions support needs to be added to the compiler tool. It is only available when using the Maven plugin ATM. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
37. scatrin commented on issue #118: AVRO-1891: Fix specific nested logical types URL: https://github.com/apache/avro/pull/118#issuecomment-414774965 I submitted a new PR today, #329, with a different solution approach to AVRO-1891. Feedback appreciated! ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
38. scatrin commented on issue #118: AVRO-1891: Fix specific nested logical types URL: https://github.com/apache/avro/pull/118#issuecomment-414942074 Hey @lewisdawson, there is a workaround for the current version that might help you. It involves implementing your own Avro serializer (or slightly modifying the one you're using, I did that with Confluent's). The key is to explicitly add the right conversions for logical types to SpecificData when creating the DatumWriter and DatumReader. If you still need help, create a post on StackOverflow and we'll take it over there. ---------- This is an

automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org

39. ivangreene commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-415516402 Very interested in this PR, I've been running into this issue as well. I'm going to try testing it but have you attempted to use this with custom defined types that use a logicalType underneath? For example: ```json [ { "type": "fixed", "namespace": "com.example.avro", "name": "SomeDecimal", "logicalType": "decimal", "scale": 4, "precision": 10, "size": 8 } ] ``` ```json { "fields": [ { "name": "foo", "type": ["null", "com.example.avro.SomeDecimal"], "default": null } ] } ``` ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
40. scatrin commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-415683420 @ivangreene As I understand it, the question is about using a generated Avro type as the target type of a custom conversion? So whenever you use the logical type "decimal" you would get that generated class as the type of your field? The target type of the conversion needs to be in the classpath of the Conversion implementation. And the Conversion implementation needs to be in the classpath of the Maven plugin. (See the integration tests for an example of this.) So it would be hard to do with both Avro types being generated at the same time without doing some serious classloading magic in the Maven plugin. However, you could have the project containing the conversion generate its target type from an Avro schema, but that doesn't make much sense to me. It should be pretty straightforward to add tests with new types of schemas in the codegen-test project if you want to try something out. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
41. kgalieva commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-426582878 @scatrin thank you for this patch! I pulled, rebased and tested it with integration with #309 Everything works just fine. What do you think about rebasing and proceeding with code review? Thanks again, this is a super useful fix of a common problem. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
42. ivangreene commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-426641789 +1 to that. Would really like to see this in the next release of Avro as we're currently using a backport of #118 onto 1.8.2 and want to get back onto an official release ASAP ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
43. kgalieva edited a comment on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-426582878 @scatrin thank you for this patch! I pulled, rebased and tested it in integration with #309 Everything works just fine. What do you think about rebasing and proceeding with code review? Thanks again, this is a super useful fix of a common problem. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
44. scatrin commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-426912754 Glad you like it @kgalieva and @ivangreene. I'll hopefully have time to rebase it during today. Is there something else I need to do for it to proceed to code review? ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
45. kgalieva commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-427035082 > I'll hopefully have time to rebase it during today. Is there something else I need to do for it to proceed to code review? @nandorKollar Could you please help to assign reviewers to this PR? ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
46. kgalieva edited a comment on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-427035082 > I'll hopefully have time to rebase it during today. Is there something else I need to do for it to proceed to code review? @nandorKollar Could you please help to assign reviewers to this PR? ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
47. scatrin commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-427566815 Hi @kgalieva, @nandorKollar, just letting you know that I'm done with the rebase. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
48. kgalieva commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-433000190 Hi @scatrin. Sorry for slow response. I've tested your rebased PR on my project. All conversions including JSR310, decimal and custom ones work just fine. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
49. kgalieva commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-434707378 Hi @scatrin! I've came across one failing case. `getUsedConversionClasses` method in `SpecificCompiler` doesn't include conversions for nested records. I define a record with logical types in one avro file. Then use those record as filed types in another avro file. When compiled, parent record doesn't contain conversions necessary for nested record serialisation. Could you please take a look? ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
50. scatrin commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-434988845 Thanks @kgalieva, I'll take a look during the weekend. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
51. scatrin commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-437316951 @kgalieva The problem should be fixed now. I see that there are conflicts to resolve too, will do another rebase soon. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
52. malcolmrobbins commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-438455726 Any chance of getting a PATCH for the collective changes so the "impatient" can apply for themselves until it comes out in an official release? ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
53. kgalieva commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-438479297 Hi @scatrin and @rstata! Thank you both for your great contributions! I've tested this build on my schemas and noticed a small bug in master that has been merged to this PR as well. The problem is that `customEncode` and `customDecode` methods are `protected`.
https://github.com/apache/avro/blame/master/lang/java/compiler/src/main/velocity/org/apache/avro/compiler/specific/templates/java/classic/record.vm#L504
https://github.com/apache/avro/blame/master/lang/java/compiler/src/main/velocity/org/apache/avro/compiler/specific/templates/java/classic/record.vm#L518 This means that record with record field located in different package won't compile. What do you think about making those methods `public`? Thanks again, and please correct me if I'm wrong here :) ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org

54. scatrin commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-438707755 Hi @kgalieva, @rstata, from my point of view there is no problem with making those methods public. However I haven't been involved at all in that feature so I might not have the full picture. Is this only a problem combined with this PR? If not, I guess it should be fixed on master or in a separate PR? ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
55. kgalieva commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-438763443 @scatrin @rstata I get the same compilation problem when I use current master version. It can be fixed in master and then rebased into this PR. If you want, I think I can do this. Let me know what you feel will be better. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
56. scatrin commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-439703392 @kgalieva Since the problem seems to be unrelated to this feature I think it's better to fix it on master and then merge it onto this branch. I have no idea when/if this PR will be merged. There are still no reviewers assigned to it. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
57. lagweasel commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-440520791 @scatrin / @kgalieva - I'm using JSR310 java.util.Instant field that can be nullable, and trying latest snapshot from scatrin:avro repo, but found that serializing to bytes and then deserializing those bytes causes a "java.lang.ClassCastException: java.lang.Long cannot be cast to java.time.Instant". I'm not familiar at all with the avro code, but have created a test that shows the issue. Would appreciate if you can take a look please. You can find my test (and details to reproduce it) here: https://github.com/lagweasel/avro-union-issue ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
58. scatrin commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-440732237 Thanks @lagweasel, I'll take a look. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
59. scatrin commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-440762900 @lagweasel thanks again, I found and corrected a bug that only appeared when SpecificDatumReader was created from a class instead of a schema. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
60. lagweasel commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-440843784 @scatrin thanks so much! Everything is working perfectly now. 👍 ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
61. lazyval commented on issue #118: AVRO-1891: Fix specific nested logical types URL: https://github.com/apache/avro/pull/118#issuecomment-444119371 @scatrin can you follow up on that? it seems like the problem is still ongoing ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
62. lazyval edited a comment on issue #118: AVRO-1891: Fix specific nested logical types URL: https://github.com/apache/avro/pull/118#issuecomment-444119371 @scatrin can you follow up on that? it seems like the problem is still ongoing despite the effort from Avro community ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
63. lazyval edited a comment on issue #118: AVRO-1891: Fix specific nested logical types URL: https://github.com/apache/avro/pull/118#issuecomment-444119371 @scatrin can you follow up on that? it seems like the problem is still ongoing despite the effort from Avro community I wonder if it's possible to do in user space, without building the custom version of avro library ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
64. lazyval commented on issue #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329#issuecomment-444124761 @scatrin seems like you would need to make a rebase to get this going ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
65. scatrin commented on issue #118: AVRO-1891: Fix specific nested logical types URL: https://github.com/apache/avro/pull/118#issuecomment-444798975 @lazyval The workaround that I did applies to a setting with Kafka and Confluent's Schema Registry. The trick is to modify the serializer/deserializers (from Confluent) to explicitly register the needed conversions when creating the SpecificDatumWriter/Reader. (for instance [here] (https://github.com/confluentinc/schema-registry/blob/ce5132bef1f6f49a7a174183a6867afbb8d564b8/avro-serializer/src/main/java/io/confluent/kafka/serializers/AbstractKafkaAvroSerializer.java#L97). SpecificDatumWriter has a [getter] (https://github.com/apache/avro/blob/b4ede4b116b24b5308e8419504a73e02b7f7e406/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificDatumReader.ja for the SpecificData instance and SpecificData (via its superclass) has a [method] (https://github.com/apache/avro/blob/bbea1cec3ed6ffae631b4c0d975639a098687ab4/lang/java/avro/src/main/java/org/apache/avro/generic/GenericData.java#L114) for registering logical type conversions. The cool thing is that the conversion class does not have to be part of Avro, you can build your own if you want to. Then just make sure that the modified classes are in the classpath when serializing/deserializing and that it is referenced properly in the Kafka settings. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
66. scatrin edited a comment on issue #118: AVRO-1891: Fix specific nested logical types URL: https://github.com/apache/avro/pull/118#issuecomment-444798975 @lazyval The workaround that I did applies to a setting with Kafka and Confluent's Schema Registry. The trick is to modify the serializer/deserializers (from Confluent) to explicitly register the needed conversions when creating the SpecificDatumWriter/Reader. (for instance [here]) (https://github.com/confluentinc/schema-registry/blob/ce5132bef1f6f49a7a174183a6867afbb8d564b8/avro-serializer/src/main/java/io/confluent/kafka/serializers/AbstractKafkaAvroSerializer.java#L97). SpecificDatumWriter has a [getter] (https://github.com/apache/avro/blob/b4ede4b116b24b5308e8419504a73e02b7f7e406/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificDatumReader.ja for the SpecificData instance and SpecificData (via its superclass) has a [method] (https://github.com/apache/avro/blob/bbea1cec3ed6ffae631b4c0d975639a098687ab4/lang/java/avro/src/main/java/org/apache/avro/generic/GenericData.java#L114) for registering logical type conversions. The cool thing is that the conversion class does not have to be part of Avro, you can build your own if you want to. Then just make sure that the modified classes are in the classpath when serializing/deserializing and that it is referenced properly in the Kafka settings. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
67. scatrin edited a comment on issue #118: AVRO-1891: Fix specific nested logical types URL: https://github.com/apache/avro/pull/118#issuecomment-444798975 @lazyval The workaround that I did applies to a setting with Kafka and Confluent's Schema Registry. The trick is to modify the serializer/deserializers (from Confluent) to explicitly register the needed conversions when creating the SpecificDatumWriter/Reader. (for instance [here] (https://github.com/confluentinc/schema-registry/blob/ce5132bef1f6f49a7a174183a6867afbb8d564b8/avro-serializer/src/main/java/io/confluent/kafka/serializers/AbstractKafkaAvroSerializer.java#L97)). SpecificDatumWriter has a [getter] (https://github.com/apache/avro/blob/b4ede4b116b24b5308e8419504a73e02b7f7e406/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificDatumReader.ja for the SpecificData instance and SpecificData (via its superclass) has a [method] (https://github.com/apache/avro/blob/bbea1cec3ed6ffae631b4c0d975639a098687ab4/lang/java/avro/src/main/java/org/apache/avro/generic/GenericData.java#L114) for registering logical type conversions. The cool thing is that the conversion class does not have to be part of Avro, you can build your own if you want to. Then just make sure that the modified classes are in the classpath when serializing/deserializing and that it is referenced properly in the Kafka settings. ----------

---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org

68. dkulp closed pull request #329: Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] URL: https://github.com/apache/avro/pull/329 This is a PR merged from a forked repository. As GitHub hides the original diff on merge, it is displayed below for the sake of provenance: As this is a foreign pull request (from a fork), the diff is supplied below (as it won't show otherwise due to GitHub magic): diff --git a/lang/java/avro/src/main/java/org/apache/avro/data/RecordBuilderBase.java b/lang/java/avro/src/main/java/org/apache/avro/data/RecordBuilderBase.java index 106c500b4..6d2f4c19e 100644 --- a/lang/java/avro/src/main/java/org/apache/avro/data/RecordBuilderBase.java +++ b/lang/java/avro/src/main/java/org/apache/avro/data/RecordBuilderBase.java @@ -17,19 +17,16 @@ */ package org.apache.avro.data; -import java.io.IOException; -import java.util.Arrays; - import org.apache.avro.AvroRuntimeException; -import org.apache.avro.Conversion; -import org.apache.avro.Conversions; -import org.apache.avro.LogicalType; import org.apache.avro.Schema; import org.apache.avro.Schema.Field; import org.apache.avro.Schema.Type; import org.apache.avro.generic.GenericData; import org.apache.avro.generic.IndexedRecord; +import java.io.IOException; +import java.util.Arrays; + /** Abstract base class for RecordBuilder implementations. Not thread-safe. */ public abstract class RecordBuilderBase<T extends IndexedRecord> implements RecordBuilder<T> { @@ -138,29 +135,6 @@ protected Object defaultValue(Field field) throws IOException { return data.deepCopy(field.schema(), data.getDefaultValue(field)); } - /** - * Gets the default value of the given field, if any. Pass in a conversion - * to convert data to logical type class. Please make sure the schema does - * have a logical type, otherwise an exception would be thrown out. - * @param field the field whose default value should be retrieved. - * @param conversion the tool to convert data to logical type class - * @return the default value associated with the given field, - * or null if none is specified in the schema. - * @throws IOException - */ - @SuppressWarnings({ "rawtypes", "unchecked" }) - protected Object defaultValue(Field field, Conversion<?> conversion) throws IOException { - Schema schema = field.schema(); - LogicalType logicalType = schema.getLogicalType(); - Object rawDefaultValue = data.deepCopy(schema, data.getDefaultValue(field)); - if (conversion == null || logicalType == null) { - return rawDefaultValue; - } else { - return Conversions.convertToLogicalType(rawDefaultValue, schema, - logicalType, conversion); - } - } - @Override public int hashCode() { final int prime = 31; diff --git a/lang/java/avro/src/main/java/org/apache/avro/generic/GenericData.java b/lang/java/avro/src/main/java/org/apache/avro/generic/GenericData.java index 6dffa15c5..7294192f3 100644 --- a/lang/java/avro/src/main/java/org/apache/avro/generic/GenericData.java +++ b/lang/java/avro/src/main/java/org/apache/avro/generic/GenericData.java @@ -105,6 +105,10 @@ public GenericData(ClassLoader classLoader) { private Map<Class<?>, Map<String, Conversion<?>>> conversionsByClass = new IdentityHashMap<>(); + public Collection<Conversion<?>> getConversions() { + return conversions.values(); + } + /** * Registers the given conversion to be used when reading and writing with * this data model. diff --git a/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificData.java b/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificData.java index d7b2bf825..c4388caaa 100644 --- a/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificData.java +++ b/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificData.java @@ -17,6 +17,7 @@ */ package org.apache.avro.specific; +import java.lang.reflect.Field; import java.util.Arrays; import java.util.HashSet; import java.util.Map; @@ -132,6 +133,55 @@ public DatumWriter createDatumWriter(Schema schema) { /** Return the singleton instance. */ public static SpecificData get() { return INSTANCE; } + /** + * For RECORD type schemas, this method returns the SpecificData instance of the class associated with the schema, + * in order to get the right conversions for any logical types used. + * + * @param reader the reader schema + * @return the SpecificData associated with the schema's class, or the default instance. + */ + public static SpecificData getForSchema(Schema reader) { + if (reader.getType() == Type.RECORD) { + final String className = getClassName(reader); + if (className != null) { + final Class<?> clazz; + try { + clazz = Class.forName(className); + return getForClass(clazz); + } catch (ClassNotFoundException e) { + return SpecificData.get(); + } + } + } + return SpecificData.get(); + } + + /** + * If the given class is assignable to {@link SpecificRecordBase}, this method returns the SpecificData instance + * from the field {@code MODEL$}, in order to get the correct {@link org.apache.avro.Conversion} instances for the class. + * Falls back to the default instance {@link SpecificData#get()} for other classes or if the field is not found. + * + * @param c A class + * @param <T> . + * @return The SpecificData from the SpecificRecordBase instance, or the default SpecificData instance. + */ + public static <T> SpecificData getForClass(Class<T> c) { + if (SpecificRecordBase.class.isAssignableFrom(c)) { + final Field specificDataField; + try { + specificDataField = c.getDeclaredField("MODEL$"); + specificDataField.setAccessible(true); + return (SpecificData) specificDataField.get(null); + } catch (NoSuchFieldException e) { + // Return default instance + return SpecificData.get(); + } catch (IllegalAccessException e) { + throw new AvroRuntimeException(e); + } + } + return SpecificData.get(); + } + private boolean useCustomCoderFlag = Boolean.parseBoolean(System.getProperty("org.apache.avro.specific.use_custom_coders","false")); diff --git a/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificDatumReader.java b/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificDatumReader.java index ccf8107ac..7fc91df30 100644 --- a/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificDatumReader.java +++ b/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificDatumReader.java @@ -33,18 +33,18 @@ public SpecificDatumReader() { /** Construct for reading instances of a class. */ public SpecificDatumReader(Class<T> c) { - this(new SpecificData(c.getClassLoader())); + this(SpecificData.getForClass(c)); setSchema(getSpecificData().getSchema(c)); } /** Construct where the writer's and reader's schemas are the same. */ public SpecificDatumReader(Schema schema) { - this(schema, schema, SpecificData.get()); + this(schema, schema, SpecificData.getForSchema(schema)); } /** Construct given writer's and reader's schema. */ public SpecificDatumReader(Schema writer, Schema reader) { - this(writer, reader, SpecificData.get()); + this(writer, reader, SpecificData.getForSchema(reader)); } /** Construct given writer's schema, reader's schema, and a {@link diff --git a/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificDatumWriter.java b/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificDatumWriter.java index 3d5e7ff4f..e4662be09 100644 --- a/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificDatumWriter.java +++ b/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificDatumWriter.java @@ -32,11 +32,11 @@ public SpecificDatumWriter() { } public SpecificDatumWriter(Class<T> c) { - super(SpecificData.get().getSchema(c), SpecificData.get()); + super(SpecificData.get().getSchema(c), SpecificData.getForClass(c)); } public SpecificDatumWriter(Schema schema) { - super(schema, SpecificData.get()); + super(schema, SpecificData.getForSchema(schema)); } public SpecificDatumWriter(Schema root, SpecificData specificData) { diff --git a/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificRecordBase.java b/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificRecordBase.java index eed41b514..ac003bacd 100644 --- a/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificRecordBase.java +++ b/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificRecordBase.java @@ -37,6 +37,11 @@ public abstract Object get(int field); public abstract void put(int field, Object value); + public SpecificData getSpecificData() { + // Default implementation for backwards compatibility, overridden in generated code + return SpecificData.get(); + } + public Conversion<?> getConversion(int field) { // for backward-compatibility. no older specific classes have conversions. return null; @@ -61,22 +66,22 @@ public boolean equals(Object that) { if (that == this) return true; // identical object if (!(that instanceof SpecificRecord)) return false; // not a record if (this.getClass() != that.getClass()) return false; // not same schema - return SpecificData.get().compare(this, that, this.getSchema(), true) == 0; + return getSpecificData().compare(this, that, this.getSchema(), true) == 0; } @Override public int hashCode() { - return SpecificData.get().hashCode(this, this.getSchema()); + return getSpecificData().hashCode(this, this.getSchema()); } @Override public int compareTo(SpecificRecord that) { - return SpecificData.get().compare(this, that, this.getSchema()); + return getSpecificData().compare(this, that, this.getSchema()); } @Override public String toString() { - return SpecificData.get().toString(this); + return getSpecificData().toString(this); } @Override diff --git a/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificRecordBuilderBase.java b/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificRecordBuilderBase.java index ecf3c34dc..a8d220b50 100644 --- a/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificRecordBuilderBase.java +++ b/lang/java/avro/src/main/java/org/apache/avro/specific/SpecificRecordBuilderBase.java @@ -32,7 +32,7 @@ * @param schema the schema associated with the record class. */ protected SpecificRecordBuilderBase(Schema schema) { - super(schema, SpecificData.get()); + super(schema, SpecificData.getForSchema(schema)); } /** @@ -40,7 +40,7 @@ protected SpecificRecordBuilderBase(Schema schema) { * @param other SpecificRecordBuilderBase instance to copy. */ protected SpecificRecordBuilderBase(SpecificRecordBuilderBase<T> other) { - super(other, SpecificData.get()); + super(other, other.data()); } /** @@ -48,6 +48,6 @@ protected SpecificRecordBuilderBase(SpecificRecordBuilderBase<T> other) { * @param other the record instance to copy. */ protected SpecificRecordBuilderBase(T other) { - super(other.getSchema(), SpecificData.get()); + super(other.getSchema(), SpecificData.getForSchema(other.getSchema())); } } diff --git a/lang/java/avro/src/test/java/org/apache/avro/specific/TestRecordWithJsr310LogicalTypes.java b/lang/java/avro/src/test/java/org/apache/avro/specific/TestRecordWithJsr310LogicalTypes.java index 56e31f4b7..352e5f0d6 100644 --- a/lang/java/avro/src/test/java/org/apache/avro/specific/TestRecordWithJsr310LogicalTypes.java +++ b/lang/java/avro/src/test/java/org/apache/avro/specific/TestRecordWithJsr310LogicalTypes.java @@ -854,16 +854,16 @@ public boolean hasDec() { public TestRecordWithJsr310LogicalTypes build() { try { TestRecordWithJsr310LogicalTypes record = new TestRecordWithJsr310LogicalTypes(); - record.b = fieldSetFlags()[0] ? this.b : (java.lang.Boolean) defaultValue(fields()[0], record.getConversion(0)); - record.i32 = fieldSetFlags()[1] ? this.i32 : (java.lang.Integer) defaultValue(fields()[1], record.getConversion(1)); - record.i64 = fieldSetFlags()[2] ? this.i64 : (java.lang.Long) defaultValue(fields()[2], record.getConversion(2)); - record.f32 = fieldSetFlags()[3] ? this.f32 : (java.lang.Float) defaultValue(fields()[3], record.getConversion(3)); - record.f64 = fieldSetFlags()[4] ? this.f64 : (java.lang.Double) defaultValue(fields()[4], record.getConversion(4)); - record.s = fieldSetFlags()[5] ? this.s : (java.lang.CharSequence) defaultValue(fields()[5], record.getConversion(5)); - record.d = fieldSetFlags()[6] ? this.d : (java.time.LocalDate) defaultValue(fields()[6], record.getConversion(6)); - record.t =

```
fieldSetFlags()[7] ? this.t : (java.time.LocalTime) defaultValue(fields()[7], record.getConversion(7)); - record.ts = fieldSetFlags()[8] ? this.ts : (java.time.Instant) defaultValue(fields()[8], record.getConversion(8)); - record.dec = fieldSetFlags()[9] ? this.dec : (java.math.BigDecimal) defaultValue(fields()[9], record.getConversion(9)); + record.b = fieldSetFlags()[0] ? this.b : (java.lang.Boolean) defaultValue(fields()[0]); + record.i32 = fieldSetFlags()[1] ? this.i32 : (java.lang.Integer) defaultValue(fields()[1]); + record.i64 = fieldSetFlags()[2] ? this.i64 : (java.lang.Long) defaultValue(fields()[2]); + record.f32 = fieldSetFlags()[3] ? this.f32 : (java.lang.Float) defaultValue(fields()[3]); + record.f64 = fieldSetFlags()[4] ? this.f64 : (java.lang.Double) defaultValue(fields()[4]); + record.s = fieldSetFlags()[5] ? this.s : (java.lang.CharSequence) defaultValue(fields()[5]); + record.d = fieldSetFlags()[6] ? this.d : (java.time.LocalDate) defaultValue(fields()[6]); + record.t = fieldSetFlags()[7] ? this.t : (java.time.LocalTime) defaultValue(fields()[7]); + record.ts = fieldSetFlags()[8] ? this.ts : (java.time.Instant) defaultValue(fields()[8]); + record.dec = fieldSetFlags()[9] ? this.dec : (java.math.BigDecimal) defaultValue(fields()[9]); return record; } catch (java.lang.Exception e) { throw new org.apache.avro.AvroRuntimeException(e); diff --git a/lang/java/compiler/src/main/java/org/apache/avro/compiler/specific/SpecificCompiler.java b/lang/java/compiler/src/main/java/org/apache/avro/compiler/specific/SpecificCompiler.java index b92025f61..1936bd67a 100644 --- a/lang/java/compiler/src/main/java/org/apache/avro/compiler/specific/SpecificCompiler.java +++ b/lang/java/compiler/src/main/java/org/apache/avro/compiler/specific/SpecificCompiler.java @@ -293,6 +293,72 @@ public DateTimeLogicalTypeImplementation getDateTimeLogicalTypeImplementation() return dateTimeLogicalTypeImplementation; } + public void addCustomConversion(Class<?> conversionClass) { + try { + final Conversion<?> conversion = (Conversion<?>)conversionClass.newInstance(); + specificData.addLogicalTypeConversion(conversion); + } catch (IllegalAccessException | InstantiationException e) { + throw new RuntimeException("Failed to instantiate conversion class " + conversionClass, e); + } + } + public Collection<String> getUsedConversionClasses(Schema schema) { + LinkedHashMap<String, Conversion<?>> classnameToConversion = new LinkedHashMap<>(); + for (Conversion<?> conversion : specificData.getConversions()) { + classnameToConversion.put(conversion.getConvertedType().getCanonicalName(), conversion); + } + Collection<String> result = new HashSet<>(); + for (String className : getClassNamesOfPrimitiveFields(schema)) { + if (classnameToConversion.containsKey(className)) { + result.add(classnameToConversion.get(className).getClass().getCanonicalName()); + } + } + return result; + } + private Set<String> getClassNamesOfPrimitiveFields(Schema schema) { + Set<String> result = new HashSet<>(); + getClassNamesOfPrimitiveFields(schema, result, new HashSet<>()); + return result; + } + private void getClassNamesOfPrimitiveFields(Schema schema, Set<String> result, Set<Schema> seenSchemas) { + if (seenSchemas.contains(schema)) { + return; + } + seenSchemas.add(schema); + switch (schema.getType()) { + case RECORD: + for (Schema.Field field : schema.getFields()) { + getClassNamesOfPrimitiveFields(field.schema(), result, seenSchemas); + } + break; + case MAP: + getClassNamesOfPrimitiveFields(schema.getValueType(), result, seenSchemas); + break; + case ARRAY: + getClassNamesOfPrimitiveFields(schema.getElementType(), result, seenSchemas); + break; + case UNION: + for (Schema s : schema.getTypes()) + getClassNamesOfPrimitiveFields(s, result, seenSchemas); + break; + case ENUM: + case FIXED: + case NULL: + break; + case STRING: case BYTES: + case INT: case LONG: + case FLOAT: case DOUBLE: + case BOOLEAN: + result.add(javaType(schema)); + break; + default: throw new RuntimeException("Unknown type: "+schema); + } + } + private static String logChuteName = null; + private void initializeVelocity() { this.velocityEngine = new VelocityEngine(); @@ -810,14 +876,13 @@ public String conversionInstance(Schema schema) { return "null"; } - if (LogicalTypes.date().equals(schema.getLogicalType())) { - return "DATE_CONVERSION"; - } else if (LogicalTypes.timeMillis().equals(schema.getLogicalType())) { - return "TIME_CONVERSION"; - } else if (LogicalTypes.timestampMillis().equals(schema.getLogicalType())) { - return "TIMESTAMP_CONVERSION"; - } else if (LogicalTypes.Decimal.class.equals(schema.getLogicalType().getClass())) { - return enableDecimalLogicalType ? "DECIMAL_CONVERSION" : "null"; + if (LogicalTypes.Decimal.class.equals(schema.getLogicalType().getClass()) && !enableDecimalLogicalType) { + return "null"; + } + final Conversion<Object> conversion = specificData.getConversionFor(schema.getLogicalType()); + if (conversion != null) { + return "new " + conversion.getClass().getCanonicalName() + "()"; } return "null"; diff --git a/lang/java/compiler/src/main/velocity/org/apache/avro/compiler/specific/templates/java/classic/record.vm b/lang/java/compiler/src/main/velocity/org/apache/avro/compiler/specific/templates/java/classic/record.vm index 25b4101a6..b5ee4c408 100644 --- a/lang/java/compiler/src/main/velocity/org/apache/avro/compiler/specific/templates/java/classic/record.vm +++ b/lang/java/compiler/src/main/velocity/org/apache/avro/compiler/specific/templates/java/classic/record.vm @@ -42,6 +42,14 @@ public class ${this.mangle($schema.getName())}#if ($schema.isError()) extends or public static org.apache.avro.Schema getClassSchema() { return SCHEMA$; } private static SpecificData MODEL$ = new SpecificData(); +#set ($usedConversions = $this.getUsedConversionClasses($schema)) +#if (!$usedConversions.isEmpty()) +static { +#foreach ($conversion in $usedConversions) + MODEL$.addLogicalTypeConversion(new ${conversion}()); +#end + } +#end #if (!$schema.isError()) private static final BinaryMessageEncoder<${this.mangle($schema.getName())}> ENCODER = @@ -158,6 +166,7 @@ public class ${this.mangle($schema.getName())}#if ($schema.isError()) extends or #end #end + public org.apache.avro.specific.SpecificData getSpecificData() { return MODEL$; } public org.apache.avro.Schema getSchema() { return SCHEMA$; } // Used by DatumWriter. Applications should not call. public java.lang.Object get(int field$) { @@ -172,17 +181,6 @@ public class ${this.mangle($schema.getName())}#if ($schema.isError()) extends or } #if ($this.hasLogicalTypeField($schema)) - protected static final org.apache.avro.Conversions.DecimalConversion DECIMAL_CONVERSION = new org.apache.avro.Conversions.DecimalConversion(); -#if ($this.getDateTimeLogicalTypeImplementation().name() == "JODA") - protected static final org.apache.avro.data.TimeConversions.DateConversion DATE_CONVERSION = new org.apache.avro.data.TimeConversions.DateConversion(); - protected static final org.apache.avro.data.TimeConversions.TimeConversion TIME_CONVERSION = new org.apache.avro.data.TimeConversions.TimeConversion(); - protected static final org.apache.avro.data.TimeConversions.TimestampConversion TIMESTAMP_CONVERSION = new org.apache.avro.data.TimeConversions.TimestampConversion(); -#elseif ($this.getDateTimeLogicalTypeImplementation().name() == "JSR310") - protected static final org.apache.avro.data.Jsr310TimeConversions.DateConversion DATE_CONVERSION = new org.apache.avro.data.Jsr310TimeConversions.DateConversion(); - protected static final org.apache.avro.data.Jsr310TimeConversions.TimeMillisConversion TIME_CONVERSION = new org.apache.avro.data.Jsr310TimeConversions.TimeMillisConversion(); - protected static final org.apache.avro.data.Jsr310TimeConversions.TimestampMillisConversion TIMESTAMP_CONVERSION = new org.apache.avro.data.Jsr310TimeConversions.TimestampMillisConversion(); -#end - private static final org.apache.avro.Conversion<?>[] conversions = new org.apache.avro.Conversion<?>[] { #foreach ($field in $schema.getFields()) @@ -502,19 +500,11 @@ public class ${this.mangle($schema.getName())}#if ($schema.isError()) extends or throw e; } } else { -#if ($this.hasLogicalTypeField($schema)) - record.${this.mangle($field.name(), $schema.isError())} = fieldSetFlags()[$field.pos()] ? this.${this.mangle($field.name(), $schema.isError())} : #if(${this.javaType($field.schema())} != "java.lang.Object") (${this.javaType($field.schema())})#{end} defaultValue(fields()[$field.pos()], record.getConversion($field.pos())); -#else record.${this.mangle($field.name(), $schema.isError())} = fieldSetFlags()[$field.pos()] ? this.${this.mangle($field.name(), $schema.isError())} : #if(${this.javaType($field.schema())} != "java.lang.Object") (${this.javaType($field.schema())})#{end} defaultValue(fields()[$field.pos()]); -#end } -#else -#if ($this.hasLogicalTypeField($schema)) - record.${this.mangle($field.name(), $schema.isError())} = fieldSetFlags()[$field.pos()] ? this.${this.mangle($field.name(), $schema.isError())} : #if(${this.javaType($field.schema())} != "java.lang.Object") (${this.javaType($field.schema())})#{end} defaultValue(fields()[$field.pos()], record.getConversion($field.pos())); #else record.${this.mangle($field.name(), $schema.isError())} = fieldSetFlags()[$field.pos()] ? this.${this.mangle($field.name(), $schema.isError())} : #if(${this.javaType($field.schema())} != "java.lang.Object") (${this.javaType($field.schema())})#{end} defaultValue(fields()[$field.pos()]); #end -#end #end return record; } catch (org.apache.avro.AvroMissingFieldException e) { diff --git a/lang/java/compiler/src/test/java/org/apache/avro/compiler/specific/TestSpecificCompiler.java b/lang/java/compiler/src/test/java/org/apache/avro/compiler/specific/TestSpecificCompiler.java index e1210ac21..f0cb87ab1 100644 --- a/lang/java/compiler/src/test/java/org/apache/avro/compiler/specific/TestSpecificCompiler.java +++ b/lang/java/compiler/src/test/java/org/apache/avro/compiler/specific/TestSpecificCompiler.java @@ -474,6 +474,42 @@ public void testJavaUnboxJsr310DateTime() throws Exception { "java.time.Instant", compiler.javaUnbox(timestampSchema)); } + @Test + public void testNullableLogicalTypesJavaUnboxDecimalTypesEnabled() throws Exception { + SpecificCompiler compiler = createCompiler(); + compiler.setEnableDecimalLogicalType(true); + + // Nullable types should return boxed types instead of primitive types + Schema nullableDecimalSchema1 = Schema.createUnion( + Schema.create(Schema.Type.NULL), LogicalTypes.decimal(9,2) + .addToSchema(Schema.create(Schema.Type.BYTES))); + Schema nullableDecimalSchema2 = Schema.createUnion( + LogicalTypes.decimal(9,2) + .addToSchema(Schema.create(Schema.Type.BYTES)), + Schema.create(Schema.Type.NULL)); + Assert.assertEquals("Should return boxed type", + compiler.javaUnbox(nullableDecimalSchema1), + "java.math.BigDecimal"); + Assert.assertEquals("Should return boxed type", + compiler.javaUnbox(nullableDecimalSchema2), "java.math.BigDecimal"); + } + + @Test + public void testNullableLogicalTypesJavaUnboxDecimalTypesDisabled() throws Exception { + SpecificCompiler compiler = createCompiler(); + compiler.setEnableDecimalLogicalType(false); + + // Since logical decimal types are disabled, a ByteBuffer is expected. + Schema nullableDecimalSchema1 = Schema.createUnion( + Schema.create(Schema.Type.NULL), LogicalTypes.decimal(9,2) + .addToSchema(Schema.create(Schema.Type.BYTES))); + Schema nullableDecimalSchema2 = Schema.createUnion( + LogicalTypes.decimal(9,2) + .addToSchema(Schema.create(Schema.Type.BYTES)), + Schema.create(Schema.Type.NULL)); + Assert.assertEquals("Should return boxed type", + compiler.javaUnbox(nullableDecimalSchema1), + "java.nio.ByteBuffer"); + Assert.assertEquals("Should return boxed type", + compiler.javaUnbox(nullableDecimalSchema2), "java.nio.ByteBuffer"); + } + @Test public void testNullableTypesJavaUnbox() throws Exception { SpecificCompiler compiler = createCompiler(); @@ -526,6 +562,76 @@ public void testNullableTypesJavaUnbox() throws Exception { public void
```

testNullableTypesJavaUnbox() throws Exception { compiler.javaUnbox(nullableBooleanSchema2), "java.lang.Boolean"); } + @Test + public void testGetUsedConversionClassesForNullableLogicalTypes() throws Exception { + SpecificCompiler compiler = createCompiler(); + compiler.setEnableDecimalLogicalType(true); + + Schema nullableDecimal1 = Schema.createUnion( + Schema.create(Schema.Type.NULL), LogicalTypes.decimal(9,2) + .addToSchema(Schema.create(Schema.Type.BYTES))); + Schema schemaWithNullableDecimal1 = Schema.createRecord("WithNullableDecimal", "", "", false, Collections.singletonList(new Schema.Field("decimal", nullableDecimal1, "", null))); + + final Collection<String> usedConversionClasses = compiler.getUsedConversionClasses(schemaWithNullableDecimal1); + Assert.assertEquals(1, usedConversionClasses.size()); + Assert.assertEquals("org.apache.avro.Conversions.DecimalConversion", usedConversionClasses.iterator().next()); + } + + @Test + public void testGetUsedConversionClassesForNullableLogicalTypesInNestedRecord() throws Exception { + SpecificCompiler compiler = createCompiler(); + + final Schema schema = new Schema.Parser().parse("
{\"type\":\"record\",\"name\":\"NestedLogicalTypesRecord\",\"namespace\":\"org.apache.avro.codegentest.testdata\",\"doc\":\"Test nested types with logical types in generated Java classes\",\"fields\":[{\"name\":\"nestedRecord\",\"type\":{\"type\":\"record\",\"name\":\"NestedRecord\",\"fields\":
[{\"name\":\"nullableDateField\",\"type\":[\"null\",{\"type\":\"int\",\"logicalType\":\"date\"}]}]}}]}"); + + final Collection<String> usedConversionClasses = compiler.getUsedConversionClasses(schema); + Assert.assertEquals(1, usedConversionClasses.size()); +
Assert.assertEquals("org.apache.avro.data.TimeConversions.DateConversion", usedConversionClasses.iterator().next()); + } + + @Test + public void testGetUsedConversionClassesForNullableLogicalTypesInArray() throws Exception { + SpecificCompiler compiler = createCompiler(); + + final Schema schema = new Schema.Parser().parse("{\"type\":\"record\",\"name\":\"NullableLogicalTypesArray\",\"namespace\":\"org.apache.avro.codegentest.testdata\",\"doc\":\"Test nested types with logical types in generated Java classes\",\"fields\":[{\"name\":\"arrayOfLogicalType\",\"type\":{\"type\":\"array\",\"items\":[\"null\",
{\"type\":\"int\",\"logicalType\":\"date\"}]}}]}"); + + final Collection<String> usedConversionClasses = compiler.getUsedConversionClasses(schema); + Assert.assertEquals(1, usedConversionClasses.size()); + Assert.assertEquals("org.apache.avro.data.TimeConversions.DateConversion", usedConversionClasses.iterator().next()); + } + + @Test + public void testGetUsedConversionClassesForNullableLogicalTypesInArrayOfRecords() throws Exception { + SpecificCompiler compiler = createCompiler(); + + final Schema schema = new Schema.Parser().parse("
{\"type\":\"record\",\"name\":\"NestedLogicalTypesArray\",\"namespace\":\"org.apache.avro.codegentest.testdata\",\"doc\":\"Test nested types with logical types in generated Java classes\",\"fields\":[{\"name\":\"arrayOfRecords\",\"type\":{\"type\":\"array\",\"items\":{\"type\":\"record\",\"name\":\"RecordInArray\",\"fields\":
[{\"name\":\"nullableDateField\",\"type\":[\"null\",{\"type\":\"int\",\"logicalType\":\"date\"}]}]}}}]}"); + + final Collection<String> usedConversionClasses = compiler.getUsedConversionClasses(schema); + Assert.assertEquals(1, usedConversionClasses.size()); +
Assert.assertEquals("org.apache.avro.data.TimeConversions.DateConversion", usedConversionClasses.iterator().next()); + } + + @Test + public void testGetUsedConversionClassesForNullableLogicalTypesInUnionOfRecords() throws Exception { + SpecificCompiler compiler = createCompiler(); + + final Schema schema = new Schema.Parser().parse("
{\"type\":\"record\",\"name\":\"NestedLogicalTypesUnion\",\"namespace\":\"org.apache.avro.codegentest.testdata\",\"doc\":\"Test nested types with logical types in generated Java classes\",\"fields\":[{\"name\":\"unionOfRecords\",\"type\":[\"null\",{\"type\":\"record\",\"name\":\"RecordInUnion\",\"fields\":
[{\"name\":\"nullableDateField\",\"type\":[\"null\",{\"type\":\"int\",\"logicalType\":\"date\"}]}]}]}]}"); + + final Collection<String> usedConversionClasses = compiler.getUsedConversionClasses(schema); + Assert.assertEquals(1, usedConversionClasses.size()); +
Assert.assertEquals("org.apache.avro.data.TimeConversions.DateConversion", usedConversionClasses.iterator().next()); + } + + @Test + public void testGetUsedConversionClassesForNullableLogicalTypesInMapOfRecords() throws Exception { + SpecificCompiler compiler = createCompiler(); + + final Schema schema = new Schema.Parser().parse("
{\"type\":\"record\",\"name\":\"NestedLogicalTypesMap\",\"namespace\":\"org.apache.avro.codegentest.testdata\",\"doc\":\"Test nested types with logical types in generated Java classes\",\"fields\":[{\"name\":\"mapOfRecords\",\"type\":{\"type\":\"map\",\"values\":{\"type\":\"record\",\"name\":\"RecordInMap\",\"fields\":
[{\"name\":\"nullableDateField\",\"type\":[\"null\",{\"type\":\"int\",\"logicalType\":\"date\"}]}]},\"avro.java.string\":\"String\"}}]}"); + + final Collection<String> usedConversionClasses = compiler.getUsedConversionClasses(schema); + Assert.assertEquals(1, usedConversionClasses.size()); +
Assert.assertEquals("org.apache.avro.data.TimeConversions.DateConversion", usedConversionClasses.iterator().next()); + } + @Test public void testLogicalTypesWithMultipleFields() throws Exception { Schema logicalTypesWithMultipleFields = new Schema.Parser().parse( @@ -566,12 +672,12 @@ public void testConversionInstanceWithDecimalLogicalTypeDisabled() throws Except Schema uuidSchema = LogicalTypes.uuid() .addToSchema(Schema.create(Schema.Type.STRING)); - Assert.assertEquals("Should use DATE_CONVERSION for date type", - "DATE_CONVERSION", compiler.conversionInstance(dateSchema)); - Assert.assertEquals("Should use TIME_CONVERSION for time type", - "TIME_CONVERSION", compiler.conversionInstance(timeSchema)); - Assert.assertEquals("Should use TIMESTAMP_CONVERSION for date type", - "TIMESTAMP_CONVERSION", compiler.conversionInstance(timestampSchema)); + Assert.assertEquals("Should use date conversion for date type", + "new org.apache.avro.data.TimeConversions.DateConversion()", compiler.conversionInstance(dateSchema)); + Assert.assertEquals("Should use time conversion for time type", + "new org.apache.avro.data.TimeConversions.TimeConversion()", compiler.conversionInstance(timeSchema)); + Assert.assertEquals("Should use timestamp conversion for date type", + "new org.apache.avro.data.TimeConversions.TimestampConversion()", compiler.conversionInstance(timestampSchema)); Assert.assertEquals("Should use null for decimal if the flag is off", "null", compiler.conversionInstance(decimalSchema)); Assert.assertEquals("Should use null for decimal if the flag is off", @@ -595,14 +701,14 @@ public void testConversionInstanceWithDecimalLogicalTypeEnabled() throws Excepti Schema uuidSchema = LogicalTypes.uuid() .addToSchema(Schema.create(Schema.Type.STRING)); - Assert.assertEquals("Should use DATE_CONVERSION for date type", - "DATE_CONVERSION", compiler.conversionInstance(dateSchema)); - Assert.assertEquals("Should use TIME_CONVERSION for time type", - "TIME_CONVERSION", compiler.conversionInstance(timeSchema)); - Assert.assertEquals("Should use TIMESTAMP_CONVERSION for date type", - "TIMESTAMP_CONVERSION", compiler.conversionInstance(timestampSchema)); + Assert.assertEquals("Should use date conversion for date type", + "new org.apache.avro.data.TimeConversions.DateConversion()", compiler.conversionInstance(dateSchema)); + Assert.assertEquals("Should use time conversion for time type", + "new org.apache.avro.data.TimeConversions.TimeConversion()", compiler.conversionInstance(timeSchema)); + Assert.assertEquals("Should use timestamp conversion for date type", + "new org.apache.avro.data.TimeConversions.TimestampConversion()", compiler.conversionInstance(timestampSchema)); Assert.assertEquals("Should use null for decimal if the flag is off", - "DECIMAL_CONVERSION", compiler.conversionInstance(decimalSchema)); + "new org.apache.avro.Conversions.DecimalConversion()", compiler.conversionInstance(decimalSchema)); Assert.assertEquals("Should use null for decimal if the flag is off", "null", compiler.conversionInstance(uuidSchema)); } diff --git a/lang/java/integration-test/codegen-test/pom.xml b/lang/java/integration-test/codegen-test/pom.xml new file mode 100644 index 000000000..2be8ad774 --- /dev/null +++ b/lang/java/integration-test/codegen-test/pom.xml @@ -0,0 +1,91 @@ +<?xml version="1.0" encoding="UTF-8"?> +<!-- + Licensed to the Apache Software Foundation (ASF) under one or more + contributor license agreements. See the NOTICE file distributed with + this work for additional information regarding copyright ownership. + The ASF licenses this file to You under the Apache License, Version 2.0 + (the "License"); you may not use this file except in compliance with + the License. You may obtain a copy of the License at + + http://www.apache.org/licenses/LICENSE-2.0 + + Unless required by applicable law or agreed to in writing, software + distributed under the License is distributed on an "AS IS" BASIS, + WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + See the License for the specific language governing permissions and + limitations under the License. +--> +<project + xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd" + xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> + <modelVersion>4.0.0</modelVersion> + + <parent> + <artifactId>avro-integration-test</artifactId> + <groupId>org.apache.avro</groupId> + <version>1.9.0-SNAPSHOT</version> + <relativePath>../</relativePath> + </parent> + + <artifactId>avro-codegen-test</artifactId> + + <name>Apache Avro Codegen Test</name> + <packaging>jar</packaging> + <url>http://avro.apache.org</url> + <description>Tests generated Avro Specific Java API</description> + <build> + <plugins> + <plugin> + <groupId>org.apache.avro</groupId> + <artifactId>avro-maven-plugin</artifactId> + <version>${project.version}</version> + <executions> + <execution> + <phase>generate-test-sources</phase> + <goals> + <goal>schema</goal> + <goal>protocol</goal> + <goal>idl-protocol</goal> + </goals> + <configuration> + <stringType>String</stringType> + <testSourceDirectory>${project.basedir}/src/test/resources/avro</testSourceDirectory> + <testOutputDirectory>${project.build.directory}/generated-test-sources/java</testOutputDirectory> +
<enableDecimalLogicalType>true</enableDecimalLogicalType> + <customConversions> +
<conversion>org.apache.avro.codegentest.CustomDecimalConversion</conversion> + </customConversions> + </configuration> + </execution> + </executions> + <dependencies> + <dependency> + <groupId>org.apache.avro</groupId> + <artifactId>avro-test-custom-conversions</artifactId> +
<version>${project.version}</version> + </dependency> + </dependencies> + </plugin> + </plugins> + </build> + + <dependencies> + <dependency> +
<groupId>${project.groupId}</groupId> + <artifactId>avro</artifactId> + <version>${project.version}</version> + </dependency> + <dependency> +
<groupId>${project.groupId}</groupId> + <artifactId>avro-compiler</artifactId> + <version>${project.version}</version> + </dependency> + <dependency> +
<groupId>${project.groupId}</groupId> + <artifactId>avro-test-custom-conversions</artifactId> + <version>${project.version}</version> + </dependency> +
</dependencies> + +</project> diff --git a/lang/java/integration-test/codegen-test/src/test/java/org/apache/avro/codegentest/AbstractSpecificRecordTest.java
b/lang/java/integration-test/codegen-test/src/test/java/org/apache/avro/codegentest/AbstractSpecificRecordTest.java new file mode 100644 index
000000000..9d8a273cc --- /dev/null +++ b/lang/java/integration-test/codegen-test/src/test/java/org/apache/avro/codegentest/AbstractSpecificRecordTest.java @@
-0,0 +1,73 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version

2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +package org.apache.avro.codegentest; + +import org.apache.avro.io.DecoderFactory; +import org.apache.avro.io.EncoderFactory; +import org.apache.avro.specific.SpecificDatumReader; +import org.apache.avro.specific.SpecificDatumWriter; +import org.apache.avro.specific.SpecificRecordBase; +import org.junit.Assert; + +import java.io.ByteArrayInputStream; +import java.io.ByteArrayOutputStream; +import java.io.IOException; + +abstract class AbstractSpecificRecordTest { + + @SuppressWarnings("unchecked") + <T extends SpecificRecordBase> void verifySerDeAndStandardMethods(T original) { + final SpecificDatumWriter<T> datumWriterFromSchema = new SpecificDatumWriter<>(original.getSchema()); + final SpecificDatumReader<T> datumReaderFromSchema = new SpecificDatumReader<>(original.getSchema(), original.getSchema()); + verifySerDeAndStandardMethods(original, datumWriterFromSchema, datumReaderFromSchema); + final SpecificDatumWriter<T> datumWriterFromClass = new SpecificDatumWriter(original.getClass()); + final SpecificDatumReader<T> datumReaderFromClass = new SpecificDatumReader(original.getClass()); + verifySerDeAndStandardMethods(original, datumWriterFromClass, datumReaderFromClass); + } + + private <T extends SpecificRecordBase> void verifySerDeAndStandardMethods(T original, SpecificDatumWriter<T> datumWriter, SpecificDatumReader<T> datumReader) { + final byte[] serialized = serialize(original, datumWriter); + final T copy = deserialize(serialized, datumReader); + Assert.assertEquals(original, copy); + // In addition to equals() tested above, make sure the other methods that use SpecificData work as intended + // compareTo() throws an exception for maps, otherwise we would have tested it here + // Assert.assertEquals(0, original.compareTo(copy)); + Assert.assertEquals(original.hashCode(), copy.hashCode()); + Assert.assertEquals(original.toString(), copy.toString()); + } + + private <T extends SpecificRecordBase> byte[] serialize(T object, SpecificDatumWriter<T> datumWriter) { + ByteArrayOutputStream outputStream = new ByteArrayOutputStream(); + try { + datumWriter.write(object, EncoderFactory.get().directBinaryEncoder(outputStream, null)); + return outputStream.toByteArray(); + } catch (IOException e) { + throw new RuntimeException(e); + } + } + + private <T extends SpecificRecordBase> T deserialize(byte[] bytes, SpecificDatumReader<T> datumReader) { + try { + final ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(bytes); + return datumReader.read(null, DecoderFactory.get().directBinaryDecoder(byteArrayInputStream, null)); + } catch (IOException e) { + throw new RuntimeException(e); + } + } +} diff --git a/lang/java/integration-test/codegen-test/src/test/java/org/apache/avro/codegentest/TestCustomConversion.java b/lang/java/integration-test/codegen-test/src/test/java/org/apache/avro/codegentest/TestCustomConversion.java new file mode 100644 index 000000000..55e60a224 --- /dev/null +++ b/lang/java/integration-test/codegen-test/src/test/java/org/apache/avro/codegentest/TestCustomConversion.java @@ -0,0 +1,45 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +package org.apache.avro.codegentest; + +import org.apache.avro.codegentest.testdata.LogicalTypesWithCustomConversion; +import org.junit.Test; + +import java.io.IOException; +import java.math.BigInteger; + +public class TestCustomConversion extends AbstractSpecificRecordTest { + + @Test + public void testNullValues() throws IOException { + LogicalTypesWithCustomConversion instanceOfGeneratedClass = LogicalTypesWithCustomConversion.newBuilder() + .setNonNullCustomField(new CustomDecimal(BigInteger.valueOf(100), 2)) + .build(); + verifySerDeAndStandardMethods(instanceOfGeneratedClass); + } + + @Test + public void testNonNullValues() throws IOException { + LogicalTypesWithCustomConversion instanceOfGeneratedClass = LogicalTypesWithCustomConversion.newBuilder() + .setNonNullCustomField(new CustomDecimal(BigInteger.valueOf(100), 2)) + .setNullableCustomField(new CustomDecimal(BigInteger.valueOf(3000), 2)) + .build(); + verifySerDeAndStandardMethods(instanceOfGeneratedClass); + } +} diff --git a/lang/java/integration-test/codegen-test/src/test/java/org/apache/avro/codegentest/TestLogicalTypesWithDefaults.java b/lang/java/integration-test/codegen-test/src/test/java/org/apache/avro/codegentest/TestLogicalTypesWithDefaults.java new file mode 100644 index 000000000..c2d2d6d2f --- /dev/null +++ b/lang/java/integration-test/codegen-test/src/test/java/org/apache/avro/codegentest/TestLogicalTypesWithDefaults.java @@ -0,0 +1,58 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +package org.apache.avro.codegentest; + +import org.apache.avro.codegentest.testdata.LogicalTypesWithDefaults; +import org.joda.time.LocalDate; +import org.junit.Assert; +import org.junit.Test; + +import java.io.IOException; + +public class TestLogicalTypesWithDefaults extends AbstractSpecificRecordTest { + + private static final LocalDate DEFAULT_VALUE = LocalDate.parse("1973-05-19"); + + @Test + public void testDefaultValueOfNullableField() throws IOException { + LogicalTypesWithDefaults instanceOfGeneratedClass = LogicalTypesWithDefaults.newBuilder() + .setNonNullDate(LocalDate.now()) + .build(); + verifySerDeAndStandardMethods(instanceOfGeneratedClass); + } + + @Test + public void testDefaultValueOfNonNullField() throws IOException { + LogicalTypesWithDefaults instanceOfGeneratedClass = LogicalTypesWithDefaults.newBuilder() + .setNullableDate(LocalDate.now()) + .build(); + Assert.assertEquals(DEFAULT_VALUE, instanceOfGeneratedClass.getNonNullDate()); + verifySerDeAndStandardMethods(instanceOfGeneratedClass); + } + + @Test + public void testWithValues() throws IOException { + LogicalTypesWithDefaults instanceOfGeneratedClass = LogicalTypesWithDefaults.newBuilder() + .setNullableDate(LocalDate.now()) + .setNonNullDate(LocalDate.now()) + .build(); + verifySerDeAndStandardMethods(instanceOfGeneratedClass); + } +} diff --git a/lang/java/integration-test/codegen-test/src/test/java/org/apache/avro/codegentest/TestNestedLogicalTypes.java b/lang/java/integration-test/codegen-test/src/test/java/org/apache/avro/codegentest/TestNestedLogicalTypes.java new file mode 100644 index 000000000..a33d038f0 --- /dev/null +++ b/lang/java/integration-test/codegen-test/src/test/java/org/apache/avro/codegentest/TestNestedLogicalTypes.java @@ -0,0 +1,67 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.avro.codegentest; + +import org.apache.avro.codegentest.testdata.*; +import org.joda.time.LocalDate; +import org.junit.Test; + +import java.util.Collections; + +public class TestNestedLogicalTypes extends AbstractSpecificRecordTest { + + @Test + public void testNullableLogicalTypeInNestedRecord() { + final NestedLogicalTypesRecord nestedLogicalTypesRecord = + NestedLogicalTypesRecord.newBuilder() + .setNestedRecord(NestedRecord.newBuilder() + .setNullableDateField(LocalDate.now()).build()).build(); + verifySerDeAndStandardMethods(nestedLogicalTypesRecord); + } + + @Test + public void testNullableLogicalTypeInArray() { + final NullableLogicalTypesArray logicalTypesArray = + NullableLogicalTypesArray.newBuilder().setArrayOfLogicalType(Collections.singletonList(LocalDate.now())).build(); + verifySerDeAndStandardMethods(logicalTypesArray); + } + + @Test + public void testNullableLogicalTypeInRecordInArray() { + final NestedLogicalTypesArray nestedLogicalTypesArray = + NestedLogicalTypesArray.newBuilder().setArrayOfRecords(Collections.singletonList( + RecordInArray.newBuilder().setNullableDateField(LocalDate.now()).build())).build(); + verifySerDeAndStandardMethods(nestedLogicalTypesArray); + } + + @Test + public void testNullableLogicalTypeInRecordInUnion() { + final NestedLogicalTypesUnion nestedLogicalTypesUnion = + NestedLogicalTypesUnion.newBuilder().setUnionOfRecords( + RecordInUnion.newBuilder().setNullableDateField(LocalDate.now()).build()).build(); + verifySerDeAndStandardMethods(nestedLogicalTypesUnion); + } + + @Test + public void testNullableLogicalTypeInRecordInMap() { + final NestedLogicalTypesMap nestedLogicalTypesMap = + NestedLogicalTypesMap.newBuilder().setMapOfRecords(Collections.singletonMap("key", + RecordInMap.newBuilder().setNullableDateField(LocalDate.now()).build())).build(); + verifySerDeAndStandardMethods(nestedLogicalTypesMap); + } +} diff --git a/lang/java/integration-test/codegen-test/src/test/java/org/apache/avro/codegentest/TestNullableLogicalTypes.java b/lang/java/integration-test/codegen-test/src/test/java/org/apache/avro/codegentest/TestNullableLogicalTypes.java new file mode 100644 index 000000000..3a44174dc --- /dev/null +++ b/lang/java/integration-test/codegen-test/src/test/java/org/apache/avro/codegentest/TestNullableLogicalTypes.java @@ -0,0 +1,45 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the

License. + */ + +package org.apache.avro.codegentest; + +import org.apache.avro.codegentest.testdata.NullableLogicalTypes; +import org.joda.time.LocalDate; +import org.junit.Test; + +import java.io.IOException; + +public class TestNullableLogicalTypes extends AbstractSpecificRecordTest { + + @Test + public void testWithNullValues() throws IOException { + NullableLogicalTypes instanceOfGeneratedClass = NullableLogicalTypes.newBuilder() + .setNullableDate(null) + .build(); + verifySerDeAndStandardMethods(instanceOfGeneratedClass); + } + + @Test + public void testDate() throws IOException { + NullableLogicalTypes instanceOfGeneratedClass = NullableLogicalTypes.newBuilder() + .setNullableDate(LocalDate.now()) + .build(); + verifySerDeAndStandardMethods(instanceOfGeneratedClass); + } + +} diff --git a/lang/java/integration-test/codegen-test/src/test/resources/avro/custom_conversion.avsc b/lang/java/integration-test/codegen-test/src/test/resources/avro/custom_conversion.avsc new file mode 100644 index 000000000..ff33c39fa --- /dev/null +++ b/lang/java/integration-test/codegen-test/src/test/resources/avro/custom_conversion.avsc @@ -0,0 +1,12 @@ +{"namespace": "org.apache.avro.codegentest.testdata", + "type": "record", + "name": "LogicalTypesWithCustomConversion", + "doc" : "Test unions with logical types in generated Java classes", + "fields": [ + {"name": "nullableCustomField", "type": ["null", {"type": "bytes", "logicalType": "decimal", "precision": 9, "scale": 2}], "default": null}, + {"name": "nonNullCustomField", "type": {"type": "bytes", "logicalType": "decimal", "precision": 9, "scale": 2}} + ] +} + + + diff --git a/lang/java/integration-test/codegen-test/src/test/resources/avro/logical_types_with_default_values.avsc b/lang/java/integration-test/codegen-test/src/test/resources/avro/logical_types_with_default_values.avsc new file mode 100644 index 000000000..d164b0a65 --- /dev/null +++ b/lang/java/integration-test/codegen-test/src/test/resources/avro/logical_types_with_default_values.avsc @@ -0,0 +1,12 @@ +{"namespace": "org.apache.avro.codegentest.testdata", + "type": "record", + "name": "LogicalTypesWithDefaults", + "doc" : "Test logical types and default values in generated Java classes", + "fields": [ + {"name": "nullableDate", "type": [{"type": "int", "logicalType": "date"}, "null"], "default": 1234}, + {"name": "nonNullDate", "type": {"type": "int", "logicalType": "date"}, "default": 1234} + ] +} + + + diff --git a/lang/java/integration-test/codegen-test/src/test/resources/avro/nested_logical_types_array.avsc b/lang/java/integration-test/codegen-test/src/test/resources/avro/nested_logical_types_array.avsc new file mode 100644 index 000000000..c5eba1423 --- /dev/null +++ b/lang/java/integration-test/codegen-test/src/test/resources/avro/nested_logical_types_array.avsc @@ -0,0 +1,26 @@ +{"namespace": "org.apache.avro.codegentest.testdata", + "type": "record", + "name": "NestedLogicalTypesArray", + "doc" : "Test nested types with logical types in generated Java classes", + "fields": [ + { + "name": "arrayOfRecords", + "type": { + "type": "array", + "items": { + "namespace": "org.apache.avro.codegentest.testdata", + "name": "RecordInArray", + "type": "record", + "fields": [ + { + "name": "nullableDateField", + "type": ["null", {"type": "int", "logicalType": "date"}] + } + ] + } + } + } +] +} + + + diff --git a/lang/java/integration-test/codegen-test/src/test/resources/avro/nested_logical_types_map.avsc b/lang/java/integration-test/codegen-test/src/test/resources/avro/nested_logical_types_map.avsc new file mode 100644 index 000000000..f99e457db --- /dev/null +++ b/lang/java/integration-test/codegen-test/src/test/resources/avro/nested_logical_types_map.avsc @@ -0,0 +1,26 @@ +{"namespace": "org.apache.avro.codegentest.testdata", + "type": "record", + "name": "NestedLogicalTypesMap", + "doc" : "Test nested types with logical types in generated Java classes", + "fields": [ + { + "name": "mapOfRecords", + "type": { + "type": "map", + "values": { + "namespace": "org.apache.avro.codegentest.testdata", + "name": "RecordInMap", + "type": "record", + "fields": [ + { + "name": "nullableDateField", + "type": ["null", {"type": "int", "logicalType": "date"}] + } + ] + } + } + } +] +} + + + diff --git a/lang/java/integration-test/codegen-test/src/test/resources/avro/nested_logical_types_record.avsc b/lang/java/integration-test/codegen-test/src/test/resources/avro/nested_logical_types_record.avsc new file mode 100644 index 000000000..d51ac8686 --- /dev/null +++ b/lang/java/integration-test/codegen-test/src/test/resources/avro/nested_logical_types_record.avsc @@ -0,0 +1,23 @@ +{"namespace": "org.apache.avro.codegentest.testdata", + "type": "record", + "name": "NestedLogicalTypesRecord", + "doc" : "Test nested types with logical types in generated Java classes", + "fields": [ + { + "name": "nestedRecord", + "type": { + "namespace": "org.apache.avro.codegentest.testdata", + "type": "record", + "name": "NestedRecord", + "fields": [ + { + "name": "nullableDateField", + "type": ["null", {"type": "int", "logicalType": "date"}] + } + ] + } + } +] +} + + + diff --git a/lang/java/integration-test/codegen-test/src/test/resources/avro/nested_logical_types_union.avsc b/lang/java/integration-test/codegen-test/src/test/resources/avro/nested_logical_types_union.avsc new file mode 100644 index 000000000..44a495c4a --- /dev/null +++ b/lang/java/integration-test/codegen-test/src/test/resources/avro/nested_logical_types_union.avsc @@ -0,0 +1,23 @@ +{"namespace": "org.apache.avro.codegentest.testdata", + "type": "record", + "name": "NestedLogicalTypesUnion", + "doc" : "Test nested types with logical types in generated Java classes", + "fields": [ + { + "name": "unionOfRecords", + "type": ["null", { + "namespace": "org.apache.avro.codegentest.testdata", + "name": "RecordInUnion", + "type": "record", + "fields": [ + { + "name": "nullableDateField", + "type": ["null", {"type": "int", "logicalType": "date"}] + } + ] + }] + } +] +} + + + diff --git a/lang/java/integration-test/codegen-test/src/test/resources/avro/nullable_logical_types.avsc b/lang/java/integration-test/codegen-test/src/test/resources/avro/nullable_logical_types.avsc new file mode 100644 index 000000000..0133133b4 --- /dev/null +++ b/lang/java/integration-test/codegen-test/src/test/resources/avro/nullable_logical_types.avsc @@ -0,0 +1,11 @@ +{"namespace": "org.apache.avro.codegentest.testdata", + "type": "record", + "name": "NullableLogicalTypes", + "doc" : "Test unions with logical types in generated Java classes", + "fields": [ + {"name": "nullableDate", "type": ["null", {"type": "int", "logicalType": "date"}], "default": null} + ] +} + + + diff --git a/lang/java/integration-test/codegen-test/src/test/resources/avro/nullable_logical_types_array.avsc b/lang/java/integration-test/codegen-test/src/test/resources/avro/nullable_logical_types_array.avsc new file mode 100644 index 000000000..8e5caded5 --- /dev/null +++ b/lang/java/integration-test/codegen-test/src/test/resources/avro/nullable_logical_types_array.avsc @@ -0,0 +1,16 @@ +{"namespace": "org.apache.avro.codegentest.testdata", + "type": "record", + "name": "NullableLogicalTypesArray", + "doc" : "Test nested types with logical types in generated Java classes", + "fields": [ + { + "name": "arrayOfLogicalType", + "type": { + "type": "array", + "items": ["null", {"type": "int", "logicalType": "date"}] + } + } +] +} + + + diff --git a/lang/java/integration-test/pom.xml b/lang/java/integration-test/pom.xml new file mode 100644 index 000000000..226a0dcbf --- /dev/null +++ b/lang/java/integration-test/pom.xml @@ -0,0 +1,99 @@ +<?xml version="1.0" encoding="UTF-8"?> +<!-- + Licensed to the Apache Software Foundation (ASF) under one or more + contributor license agreements. See the NOTICE file distributed with + this work for additional information regarding copyright ownership. + The ASF licenses this file to You under the Apache License, Version 2.0 + (the "License"); you may not use this file except in compliance with + the License. You may obtain a copy of the License at + + http://www.apache.org/licenses/LICENSE-2.0 + + Unless required by applicable law or agreed to in writing, software + distributed under the License is distributed on an "AS IS" BASIS, + WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + See the License for the specific language governing permissions and + limitations under the License. +--> +<project + xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd" + xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> + <modelVersion>4.0.0</modelVersion> + + <parent> + <artifactId>avro-parent</artifactId> + <groupId>org.apache.avro</groupId> + <version>1.9.0-SNAPSHOT</version> + <relativePath>../</relativePath> + </parent> + + <artifactId>avro-integration-test</artifactId> + <name>Avro Integration Tests</name> + <description>Integration tests for code generation or other things that are hard to test within the modules without creating circular Maven dependencies. </description> + <url>http://avro.apache.org/</url> + <packaging>pom</packaging> + + <modules> + <module>codegen-test</module> + <module>test-custom-conversions</module> + </modules> + + <build> + <pluginManagement> + <plugins> + <plugin> + <groupId>org.apache.maven.plugins</groupId> + <artifactId>maven-surefire-plugin</artifactId> + <version>${surefire-plugin.version}</version> + <configuration> + <failIfNoTests>false</failIfNoTests> + </configuration> + </plugin> + <plugin> + <groupId>org.apache.maven.plugins</groupId> + <artifactId>maven-compiler-plugin</artifactId> + <version>${compiler-plugin.version}</version> + <configuration> + <source>1.8</source> + <target>1.8</target> + </configuration> + </plugin> + <plugin> + <groupId>org.apache.maven.plugins</groupId> + <artifactId>maven-checkstyle-plugin</artifactId> + <version>${checkstyle-plugin.version}</version> + <configuration> + <consoleOutput>true</consoleOutput> + <configLocation>checkstyle.xml</configLocation> + </configuration> + <executions> + <execution> + <id>checkstyle-check</id> + <phase>test</phase> + <goals> + <goal>check</goal> + </goals> + </execution> + </executions> + </plugin> + <plugin> + <groupId>org.apache.maven.plugins</groupId> + <artifactId>maven-jar-plugin</artifactId> + <version>${jar-plugin.version}</version> + <executions> + <execution> + <goals> + <goal>test-jar</goal> + </goals> + </execution> + </executions> + </plugin> + </plugins> + </pluginManagement> + </build> + + <profiles> + </profiles> +</project> diff --git a/lang/java/integration-test/test-custom-conversions/pom.xml b/lang/java/integration-test/test-custom-conversions/pom.xml new file mode 100644 index 000000000..7bac7ae42 --- /dev/null +++ b/lang/java/integration-test/test-custom-conversions/pom.xml @@ -0,0 +1,45 @@ +<?xml version="1.0" encoding="UTF-8"?> +<!-- + Licensed to the Apache Software Foundation (ASF) under one or more + contributor license agreements. See the NOTICE file distributed with + this work for additional information regarding copyright ownership. + The ASF licenses this file to You under the Apache License, Version 2.0 + (the "License"); you may not use this file except in compliance with + the License. You may obtain a copy of the License at + + http://www.apache.org/licenses/LICENSE-2.0 + + Unless required by applicable law or agreed to in writing, software + distributed under the License is distributed on an "AS IS" BASIS, + WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + See the License for the specific language governing permissions and + limitations under the License. +--> +<project + xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd" + xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> + <modelVersion>4.0.0</modelVersion> + + <parent> + <artifactId>avro-integration-test</artifactId> + <groupId>org.apache.avro</groupId> + <version>1.9.0-SNAPSHOT</version> + <relativePath>../</relativePath> + </parent> + + <artifactId>avro-test-custom-conversions</artifactId> + + <name>Apache Avro Codegen Test dependencies</name> + <packaging>jar</packaging> + <url>http://avro.apache.org</url> + <description>Contains dependencies for the maven plugin used in avro-codegen-test</description> + + <dependencies> + <dependency> + <groupId>${project.groupId}</groupId> + <artifactId>avro</artifactId> + <version>${project.version}</version> + </dependency> + </dependencies> +</project> diff --git a/lang/java/integration-test/test-custom-conversions/src/main/java/org.apache.avro.codegentest/CustomDecimal.java b/lang/java/integration-test/test-custom-conversions/src/main/java/org.apache.avro.codegentest/CustomDecimal.java new file mode 100644 index 000000000..1d4f40c7e --- /dev/null +++ b/lang/java/integration-test/test-custom-conversions/src/main/java/org.apache.avro.codegentest/CustomDecimal.java @@ -0,0 +1,65 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file

information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.avro.codegentest; +import java.math.BigDecimal; +import java.math.BigInteger; + +/** + * Wraps a BigDecimal just to demonstrate that it is possible to use custom implementation classes with custom conversions. + */ +public class CustomDecimal implements Comparable<CustomDecimal> { + + private final BigDecimal internalValue; + + public CustomDecimal(BigInteger value, int scale) { + internalValue = new BigDecimal(value, scale); + } + + public byte[] toByteArray(int scale) { + final BigDecimal correctlyScaledValue; + if (scale != internalValue.scale()) { + correctlyScaledValue = internalValue.setScale(scale, BigDecimal.ROUND_HALF_UP); + } else { + correctlyScaledValue = internalValue; + } + return correctlyScaledValue.unscaledValue().toByteArray(); + } + + @Override + public boolean equals(Object o) { + if (this == o) return true; + if (o == null || getClass() != o.getClass()) return false; + + CustomDecimal that = (CustomDecimal) o; + + return internalValue.equals(that.internalValue); + } + + @Override + public int hashCode() { + return internalValue.hashCode(); + } + + @Override + public int compareTo(CustomDecimal o) { + return this.internalValue.compareTo(o.internalValue); + } +} diff --git a/lang/java/integration-test/test-custom-conversions/src/main/java/org.apache.avro.codegentest/CustomDecimalConversion.java b/lang/java/integration-test/test-custom-conversions/src/main/java/org.apache.avro.codegentest/CustomDecimalConversion.java new file mode 100644 index 000000000..7c200ad0b --- /dev/null +++ b/lang/java/integration-test/test-custom-conversions/src/main/java/org.apache.avro.codegentest/CustomDecimalConversion.java @@ -0,0 +1,52 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.avro.codegentest; +import org.apache.avro.Conversion; +import org.apache.avro.LogicalType; +import org.apache.avro.LogicalTypes; +import org.apache.avro.Schema; + +import java.math.BigInteger; +import java.nio.ByteBuffer; + +public class CustomDecimalConversion extends Conversion<CustomDecimal> { + + @Override + public Class<CustomDecimal> getConvertedType() { + return CustomDecimal.class; + } + + @Override + public String getLogicalTypeName() { + return "decimal"; + } + + public CustomDecimal fromBytes(ByteBuffer value, Schema schema, LogicalType type) { + int scale = ((LogicalTypes.Decimal)type).getScale(); + byte[] bytes = value.get(new byte[value.remaining()]).array(); + return new CustomDecimal(new BigInteger(bytes), scale); + } + + public ByteBuffer toBytes(CustomDecimal value, Schema schema, LogicalType type) { + int scale = ((LogicalTypes.Decimal)type).getScale(); + return ByteBuffer.wrap(value.toByteArray(scale)); + } +} diff --git a/lang/java/mapred/pom.xml b/lang/java/mapred/pom.xml index 4309b6709..5bffd2649 100644 --- a/lang/java/mapred/pom.xml +++ b/lang/java/mapred/pom.xml @@ -48,6 +48,18 @@ <build> <plugins> + <plugin> + <groupId>org.apache.maven.plugins</groupId> + <artifactId>maven-jar-plugin</artifactId> + <version>${jar-plugin.version}</version> + <executions> + <execution> + <goals> + <goal>test-jar</goal> + </goals> + </execution> + </executions> + </plugin> <plugin> <groupId>${project.groupId}</groupId> <artifactId>avro-maven-plugin</artifactId> diff --git a/lang/java/maven-plugin/src/main/java/org/apache/avro/mojo/AbstractAvroMojo.java b/lang/java/maven-plugin/src/main/java/org/apache/avro/mojo/AbstractAvroMojo.java index 23b77d5be..0d7fbee7a 100644 --- a/lang/java/maven-plugin/src/main/java/org/apache/avro/mojo/AbstractAvroMojo.java +++ b/lang/java/maven-plugin/src/main/java/org/apache/avro/mojo/AbstractAvroMojo.java @@ -20,10 +20,16 @@ import java.io.File; import java.io.IOException; +import java.net.MalformedURLException; +import java.net.URL; +import java.net.URLClassLoader; +import java.util.ArrayList; import java.util.Arrays; +import java.util.List; import org.apache.avro.compiler.specific.SpecificCompiler; import org.apache.avro.compiler.specific.SpecificCompiler.DateTimeLogicalTypeImplementation; import org.apache.maven.artifact.DependencyResolutionRequiredException; import org.apache.maven.plugin.AbstractMojo; import org.apache.maven.plugin.MojoExecutionException; import org.apache.maven.project.MavenProject; @@ -141,6 +147,14 @@ */ protected boolean createSetters; + /** + * A set of fully qualified class names of custom {@link org.apache.avro.Conversion} implementations to add to the compiler. + * The classes must be on the classpath at compile time and whenever the Java objects are serialized. + * + * @parameter property="customConversions" + */ + protected String[] customConversions = new String[0]; + /** * Determines whether or not to use Java classes for decimal types * @@ -282,6 +296,24 @@ protected DateTimeLogicalTypeImplementation getDateTimeLogicalTypeImplementation protected abstract void doCompile(String filename, File sourceDirectory, File outputDirectory) throws IOException; + protected URLClassLoader createClassLoader() throws DependencyResolutionRequiredException, MalformedURLException { + List<URL> urls = appendElements(project.getRuntimeClasspathElements()); + urls.addAll(appendElements(project.getTestClasspathElements())); + return new URLClassLoader(urls.toArray(new URL[urls.size()]), + Thread.currentThread().getContextClassLoader()); + } + + private List<URL> appendElements(List runtimeClasspathElements) throws MalformedURLException { + List<URL> runtimeUrls = new ArrayList<>(); + if (runtimeClasspathElements != null) { + for (Object runtimeClasspathElement : runtimeClasspathElements) { + String element = (String) runtimeClasspathElement; + runtimeUrls.add(new File(element).toURI().toURL()); + } + } + return runtimeUrls; + } + protected abstract String[] getIncludes(); protected abstract String[] getTestIncludes(); diff --git a/lang/java/maven-plugin/src/main/java/org/apache/avro/mojo/IDLProtocolMojo.java b/lang/java/maven-plugin/src/main/java/org/apache/avro/mojo/IDLProtocolMojo.java index da1ae3322..973090137 100644 --- a/lang/java/maven-plugin/src/main/java/org/apache/avro/mojo/IDLProtocolMojo.java +++ b/lang/java/maven-plugin/src/main/java/org/apache/avro/mojo/IDLProtocolMojo.java @@ -82,7 +82,6 @@ protected void doCompile(String filename, File sourceDirectory, File outputDirec URLClassLoader projPathLoader = new URLClassLoader (runtimeUrls.toArray(new URL[0]), Thread.currentThread().getContextClassLoader()); - try (Idl parser = new Idl(new File(sourceDirectory, filename), projPathLoader)) { Protocol p = parser.CompilationUnit(); @@ -96,6 +95,9 @@ protected void doCompile(String filename, File sourceDirectory, File outputDirec compiler.setGettersReturnOptional(gettersReturnOptional); compiler.setCreateSetters(createSetters); compiler.setEnableDecimalLogicalType(enableDecimalLogicalType); + for (String customConversion : customConversions) { + compiler.addCustomConversion(projPathLoader.loadClass(customConversion)); + } compiler.setOutputCharacterEncoding(project.getProperties().getProperty("project.build.sourceEncoding")); compiler.compileToDestination(null, outputDirectory); } @@ -103,6 +105,8 @@ protected void doCompile(String filename, File sourceDirectory, File outputDirec throw new IOException(drre); + } catch (ClassNotFoundException e) { + throw new IOException(e); } } diff --git a/lang/java/maven-plugin/src/main/java/org/apache/avro/mojo/ProtocolMojo.java b/lang/java/maven-plugin/src/main/java/org/apache/avro/mojo/ProtocolMojo.java index d4b3bf544..ab789031f 100644 --- a/lang/java/maven-plugin/src/main/java/org/apache/avro/mojo/ProtocolMojo.java +++ b/lang/java/maven-plugin/src/main/java/org/apache/avro/mojo/ProtocolMojo.java @@ -22,15 +22,18 @@ import java.io.File; import java.io.IOException; +import java.net.URLClassLoader; import org.apache.avro.Protocol; import org.apache.avro.compiler.specific.SpecificCompiler; +import org.apache.maven.artifact.DependencyResolutionRequiredException; /** * Generate Java classes and interfaces from Avro protocol files (.avpr) * * @goal protocol * @phase generate-sources + * @requiresDependencyResolution runtime * @threadSafe */ public class ProtocolMojo extends AbstractAvroMojo { @@ -64,17 +67,32 @@ protected void doCompile(String filename, File sourceDirectory, File outputDirec compiler.setGettersReturnOptional(gettersReturnOptional); compiler.setCreateSetters(createSetters); compiler.setEnableDecimalLogicalType(enableDecimalLogicalType); + final URLClassLoader classLoader; + try { + classLoader = createClassLoader(); + for (String customConversion : customConversions) { + compiler.addCustomConversion(classLoader.loadClass(customConversion)); + } + } catch (DependencyResolutionRequiredException e) { + throw new IOException(e); + } catch (ClassNotFoundException e) { + throw new IOException(e); + } compiler.setOutputCharacterEncoding(project.getProperties().getProperty("project.build.sourceEncoding")); compiler.compileToDestination(src, outputDirectory); } diff --git a/lang/java/maven-plugin/src/main/java/org/apache/avro/mojo/SchemaMojo.java b/lang/java/maven-plugin/src/main/java/org/apache/avro/mojo/SchemaMojo.java index 9b4840c72..55eb96a99 100644 --- a/lang/java/maven-plugin/src/main/java/org/apache/avro/mojo/SchemaMojo.java +++ b/lang/java/maven-plugin/src/main/java/org/apache/avro/mojo/SchemaMojo.java @@ -22,15 +22,18 @@ import java.io.File; import java.io.IOException; +import java.net.URLClassLoader; import org.apache.avro.Schema; import org.apache.avro.compiler.specific.SpecificCompiler; +import org.apache.maven.artifact.DependencyResolutionRequiredException; /** * Generate Java classes from Avro schema files (.avsc) * * @goal schema * @phase generate-sources + * @requiresDependencyResolution runtime+test * @threadSafe */ public class SchemaMojo extends AbstractAvroMojo { @@ -81,6 +84,16 @@ protected void doCompile(String filename, File sourceDirectory, File outputDirec compiler.setGettersReturnOptional(gettersReturnOptional); compiler.setCreateSetters(createSetters); compiler.setEnableDecimalLogicalType(enableDecimalLogicalType); + try { + final URLClassLoader classLoader = createClassLoader(); + for (String customConversion : customConversions) { + compiler.addCustomConversion(classLoader.loadClass(customConversion)); + } + } catch (ClassNotFoundException e) { + throw new IOException(e); + } catch (DependencyResolutionRequiredException e) { + throw new IOException(e); + } compiler.setOutputCharacterEncoding(project.getProperties().getProperty("project.build.sourceEncoding")); compiler.compileToDestination(src, outputDirectory); } diff --git a/lang/java/pom.xml b/lang/java/pom.xml index 773d71c5e..8e5bffae2 100644 --- a/lang/java/pom.xml +++ b/lang/java/pom.xml @@ -95,6 +95,7 @@ <module>thrift</module> <module>archetypes</module> <module>grpc</module> + <module>integration-test</module> </modules> <build> diff --git

a/lang/java/tools/src/test/compiler/output-string/avro/examples/baseball/Player.java b/lang/java/tools/src/test/compiler/output-string/avro/examples/baseball/Player.java index 531cc6fd9..691831c5a 100644 --- a/lang/java/tools/src/test/compiler/output-string/avro/examples/baseball/Player.java +++ b/lang/java/tools/src/test/compiler/output-string/avro/examples/baseball/Player.java @@ -99,6 +99,7 @@ public Player(java.lang.Integer number, java.lang.String first_name, java.lang.S this.position = position; } + public org.apache.avro.specific.SpecificData getSpecificData() { return MODEL$; } public org.apache.avro.Schema getSchema() { return SCHEMA$; } // Used by DatumWriter. Applications should not call. public java.lang.Object get(int field$) { diff --git a/lang/java/tools/src/test/compiler/output/Player.java b/lang/java/tools/src/test/compiler/output/Player.java index 94fc7d0b3..869239589 100644 --- a/lang/java/tools/src/test/compiler/output/Player.java +++ b/lang/java/tools/src/test/compiler/output/Player.java @@ -99,6 +99,7 @@ public Player(java.lang.Integer number, java.lang.CharSequence first_name, java. this.position = position; } + public org.apache.avro.specific.SpecificData getSpecificData() { return MODEL$; } public org.apache.avro.Schema getSchema() { return SCHEMA$; } // Used by DatumWriter. Applications should not call. public java.lang.Object get(int field$) { ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org

69. Commit 7ed38d7c7e150987ef8bf035196576fc158e03eb in avro's branch refs/heads/master from Katrin Skoglund [ https://gitbox.apache.org/repos/asf?p=avro.git;h=7ed38d7 ] Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] (#329) * Added end-to-end test that reproduces union with logical types problem * Adding required conversions to SpecificData in generated class (same as in SpecificCompiler) * Added test with BigDecimal * Added test with BigDecimal * Introduced customizable conversions in compiler and Maven plugin. * Fixed bug * Fixed Maven plugin classpath * Get the correct SpecificData whenever possible, to get the right conversions * No need to expose the map of conversions so expose only the values. * Better tests * Default values and conversions * Cleanup of some changes in Maven plugin * Fixed equals() for classes with nested logical types. Improved tests * Added missing copyright statement * Fixed compile error after rebase * Fixed problem with logical types in nested records. * Fixed failing test. * Fixed serialization problem when creating SpecificDatumReader from a class
70. dkulp commented on issue #118: AVRO-1891: Fix specific nested logical types URL: https://github.com/apache/avro/pull/118#issuecomment-446323747 I merged #329, is this not needed now? ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
71. Commit 7ed38d7c7e150987ef8bf035196576fc158e03eb in avro's branch refs/heads/AVRO-2238 from Katrin Skoglund [ https://gitbox.apache.org/repos/asf?p=avro.git;h=7ed38d7 ] Improved conversions handling + pluggable conversions support [AVRO-1891, AVRO-2065] (#329) * Added end-to-end test that reproduces union with logical types problem * Adding required conversions to SpecificData in generated class (same as in SpecificCompiler) * Added test with BigDecimal * Added test with BigDecimal * Introduced customizable conversions in compiler and Maven plugin. * Fixed bug * Fixed Maven plugin classpath * Get the correct SpecificData whenever possible, to get the right conversions * No need to expose the map of conversions so expose only the values. * Better tests * Default values and conversions * Cleanup of some changes in Maven plugin * Fixed equals() for classes with nested logical types. Improved tests * Added missing copyright statement * Fixed compile error after rebase * Fixed problem with logical types in nested records. * Fixed failing test. * Fixed serialization problem when creating SpecificDatumReader from a class
72. Hi, Just wanted to clarify status of this issue. Was it fixed with https://github.com/apache/avro/pull/329 ([AVRO-2065])?
73. Also curious; what are the remaining unresolved parts of this issue?
74. +1 on what the status of this issue is please.
75. Does anyone know if this is fixed in the latest `1.9.1` version? Thanks
76. I don't think folks know. The changes in PR #329 claimed to address this issue. I presume no one verified that if this issue is still open. PR#329 is in both the 1.9.0 and 1.9.1 releases, so if anyone has the time to check reproducing this issue with one of those releases it would help a bunch of folks out.
77. Hello! A quick test using the code from the description -- *this works in 1.9.1*. Hooray! The only change from the code in the description was creatint the RecordV1 using a `java.time.Instant` (due to AVRO-2335 removing the joda dateTimeLogicalTypeImpl). Hope this helps!
78. My apologies: joda-time was removed in 1.10.0, not 1.9.1. I rechecked, and the test in the description doesn't throw the exception either for `joda` or `jsr310` specific classes.
79. with 1.9.1 I am getting {code:java} Not in union ["null",{"type":"long","logicalType":"timestamp-millis"}]: 2019-10-18T03:20:49.777Z{code} Which upon debugging comes to GenericData.resolveUnion() having an empty conversionsByClass map. And following back up it looks like SpecificDatumWriter.writeField() checks for a logical type on the union itself instead of the union members. Upon not finding any logical types it then calls writeWithoutConversion(). That method resolves the union, but the logicalType conversion map is empty. I am not sure what to do about it yet. But logicalTypes in unions still seems broken at least in this code path. **update I have not gotten to the bug yet, but found that it applies to using ReflectDatumWriter(Schema). Using SpecificDatumWriter with the class or Schema constructor does not throw an exception when serializing the same datum.
80. This says it was fixed in 1.9.1, but I have the same problem (or very similar) in 1.10.0. I added comments about this to AVRO-2471 which seems to be the same but is still open.