IT SHOULD WORK, BUT IT DOESN'T

The JasperReports Cookbook

Recipes for the JasperReports Library, 6.x

> **This is one complete recipe from the book**, reproduced exactly as it ships, together with the evidence key every recipe is graded against. Nothing here is summarised or softened for the sample. If the full book disagrees with your release, it says so in the same sentence, which is the point of the lab note on the next page.

RobCoTek

What "verified" means

Every recipe ends with a **Status** line. It names the date, the fixture that produced the result, and what was actually measured. If a claim was not run, the status line says so in those words. A recipe with an untested claim in it is worth less than no recipe, because you would act on it.

The fixtures are shell scripts driving a small Java instrument against the real library. Nothing in this book is inferred from source code reading, from documentation, or from another answer.

```
JasperReports  6.20.6, pinned
Java           Temurin OpenJDK 21.0.12+8, pinned
Cross-version  6.17.0 and 7.0.0 where a claim is about version portability
```

The JVM is pinned and named because it has to be. An early run of the lab resolved `java` off `PATH` and got a patched JetBrains Runtime with its own font metrics; every verdict came out the same, but one measured overlap moved by a pixel. A book whose claim is measured behaviour cannot have a number that depends on which IDE happens to be installed.

Recipe 1. "Error loading object from file" — the message is a wrapper, read the cause

Symptom. A report that worked yesterday, or a report you have just wired up, fails at load:

```
net.sf.jasperreports.engine.JRException: Error loading object from file: invoice.jasper.
```

Nothing about the file looks wrong. It is there, it is readable, the path is right.

Why. `JRLoader` catches `IOException` and `ClassNotFoundException` from Java deserialization and wraps both in one `JRException` with one message. The message names the file. It does not name the fault. Every distinct cause in this part produces that same line, so the message alone cannot tell you anything, and reading it as though it could is what sends people to rebuild the wrong thing.

The cause chain does name the fault, and it is specific enough to act on:

```
try {
    JasperReport report = (JasperReport) JRLoader.loadObjectFromFile("invoice.jasper");
} catch (JRException e) {
    for (Throwable t = e; t != null; t = t.getCause()) {
        System.err.println(t.getClass().getName() + ": " + t.getMessage());
    }
    throw e;
}
```

Run against a `.jrxml` handed to the loader by mistake, that prints:

```
net.sf.jasperreports.engine.JRException: Error loading object from file: invoice.jrxml.
java.io.StreamCorruptedException: invalid stream header: 3C3F786D
```

`3C3F786D` is not a checksum and it is not a version marker. It is four bytes of ASCII: `3C`=`<`, `3F`=`?`, `78`=`x`, `6D`=`m`. The engine is telling you, in hex, that the file starts `<?xm` — that you handed it XML. That single line resolves the largest cause in the bucket, and it is one link down from a message that says nothing.

Fix. Always print the chain, not `e.getMessage()`. Then match the second line:

Second line	What it means	Recipe
`StreamCorruptedException: invalid stream header: 3C3F786D`	you passed the `.jrxml` to a loader that wanted `.jasper`	this one
`StreamCorruptedException`, other header bytes	the `.jasper` was rewritten in transit, usually by Maven resource filtering	this one, Watch out
`InvalidClassException: ... incompatible types for field ...`	compiled by a different major version of the library	Recipe 2
`FileNotFoundException`	the path resolved somewhere you did not expect	Recipe 4
nothing — a bare `NoClassDefFoundError` with no `JRException`	a class the engine needs at load time is missing	Recipe 7

For the `3C3F786D` case, the fix is to use the API that matches the file you have:

```
// .jrxml  -> compile it
JasperReport report = JasperCompileManager.compileReport(in);   // InputStream overload

// .jasper -> load it
JasperReport report = (JasperReport) JRLoader.loadObject(in);
```

Watch out. The same `StreamCorruptedException` with *different* header bytes is a different problem with the same shape: something rewrote the binary. The usual culprit is Maven resource filtering applied to `src/main/resources`, which does variable substitution on every file it copies and corrupts a serialized object without reporting anything. Exclude the binaries from filtering rather than moving them.

And note the reverse mistake, because it produces a message with no family resemblance to this one at all. A `.jasper` handed to the **compile** path gives:

```
JRException: org.xml.sax.SAXParseException; lineNumber: 1; columnNumber: 1;
             Invalid byte 1 of 1-byte UTF-8 sequence.
```

Two directions of one confusion, two unrelated error strings, neither of which says "you have the wrong kind of file". Recognising the pair on sight is most of this part.

Status: VERIFIED 2026-08-29 on JasperReports 6.20.6, JDK 21.0.12+8, by `verification/jasper-cookbook/grade_c01_c10.sh`. The `3C3F786D` chain was reproduced through both `JRLoader.loadObjectFromFile` and `JasperFillManager.fillReport(String, ...)`, and the four header bytes were read back off the file independently to confirm they are the file's own first bytes. The reverse-direction `SAXParseException` was reproduced as a negative control in the same run. The resource-filtering cause is reported by the mechanism but was **not** run in the lab; it is included because it produces the identical exception shape, and it is marked here rather than left to look verified.

The rest of the book

86 recipes across loading and compiling, text that will not fit, expressions and parameters, subreports, pages and breaks, styles, fonts and PDF, groups and tables, exporting, charts, datasources and beans, memory and the virtualizer, and the report query, plus an appendix of every class that moved between 6.20.6 and 7.0.0.

Every recipe ends with a Status line naming the fixture that produced its verdict and the date it ran. Where the widely repeated answer is wrong, the recipe shows the measurement. The fixtures themselves ship as the Lab Pack, so any verdict in the book can be re-run on your own bench.

robcotek.com/jasper/