

HOWTO: Compile applications with implicit import dependencies

Article ID: 000004

Last Revised On: 07-Jul-2006

The information in this article applies to:

- Excelsior JET 4.1 and below

APPLICABILITY

This article applies to Excelsior JET versions 4.1 and earlier only. Since version 4.5, all classes located in application CLASSPATH are always available to the Excelsior JET JVM either in natively compiled form or in bytecode form. It was not true for earlier JET versions and this article was written to suggest workarounds to problems that might occur because of that.

SUMMARY

In the presence of implicit class import (`Class.forName()`, etc.) the JET compiler may be unable to automatically determine all classes required by the application and include them into compilation set. This document describes the problem and shows how to diagnose and repair it.

[Download](#) sample code for this article.

MORE INFORMATION

Consider the following code snippet:

```
Object foo(String className)
{
    Class c = Class.forName(className);
    return c.newInstance();
}
```

The compiler has no way to determine which class will be created in this function. When a conventional JVM executes code like the above it can find the required class at run-time (e.g. using CLASSPATH). However, a JET-compiled executable cannot do that unless distributed with class files that may be loaded at run time and the JIT compiler. There are two ways to resolve this problem:

- Enable the JIT compiler and place required classes in CLASSPATH [??]
- Add those classes to the compilation set manually

This document describes how to add all required classes to compilation set, if they are available at compile-time.

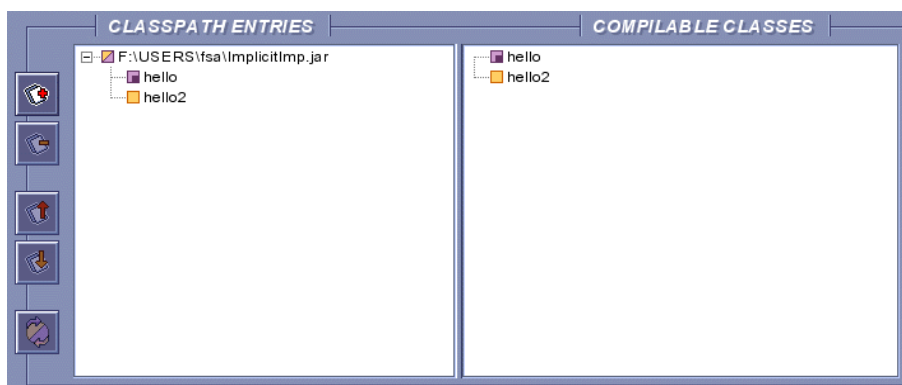
DETAILS

Example file `ImplicitImp.jar` contains two classes - `hello` and `hello2`. The main class imports `hello2` implicitly and creates its instance:

```
Class c = Class.forName("hello2");
Object o = c.newInstance();
```

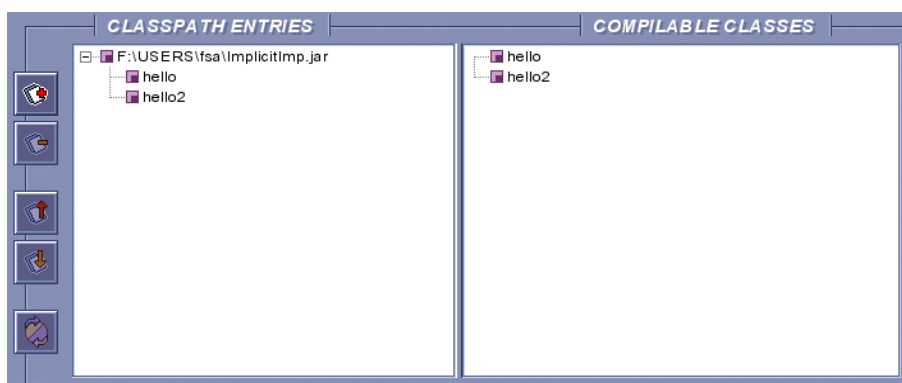
so there is no direct references to `hello2` class. If JET Control Panel is used to compile this application, it shows the user that there are some classes in classpath that are not directly imported by the application and lets the user add such classes to the compilation set or leave them alone. As shown on Figure ??, classes not included into compilation set have yellow tags.

Figure 1: `hello2` class is not included.



At this point the user can force required classes into compilation set (using right mouse button or doubleclicking on class or package names). Figure ?? illustrates how it looks if all classes are included.

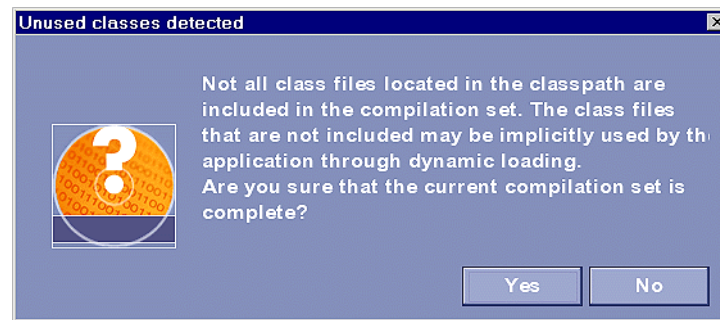
Figure 2: All classes are included into compilation set.



Now compilation may be performed and all the classes will be included into the resulting executable. When the user is sure that omitted classes are not required for application or s/he is going to use JIT compiler to load them at run-time s/he can leave them alone. In such case JET Control Panel will issue a warning about these classes before compilation (Figure ??). At this point the user can return back and enlarge compilation set or skip the warning.

Figures ??, ?? below illustrate a “real world” situation: `SimpleWrite` class imports `comm.jar` with Java Communications API [?]. Not all classes are directly imported (Figure ??) and the user should

Figure 3: Unused class reminder.



force them into compilation set manually (Figure ??), using right mouse button or doubleclicking on class or package names.

Figure 4: Java Communications API - not all classes are imported explicitly.

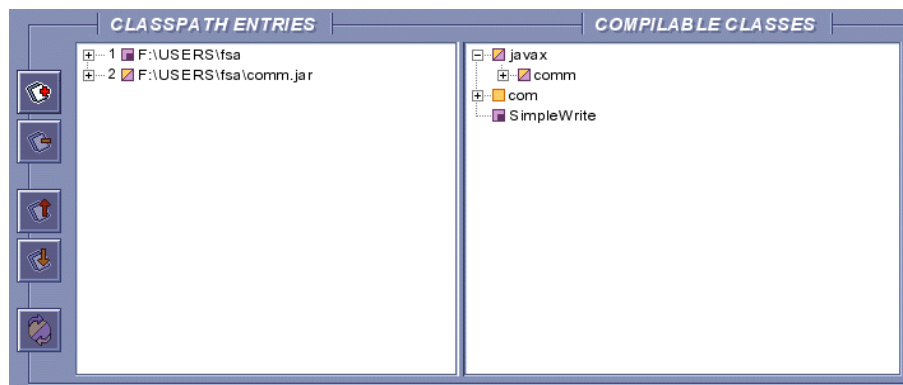
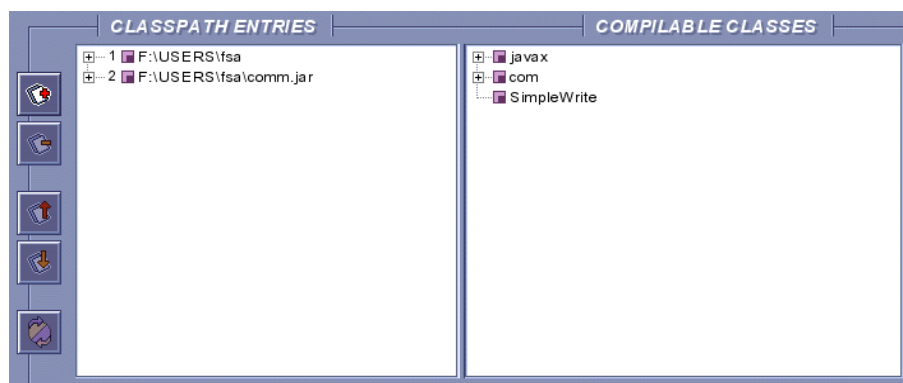


Figure 5: Java Communications API - all classes are forced into compilation set.



REFERENCES

1. The JET Control Panel: <http://www.excelsior-usa.com/jetdoc.html>
2. Mixed compilation model: <http://www.excelsior-usa.com/jetdoc.html>

3. Java Communications API: <http://java.sun.com/products/javacomm>