

CMT & Java

(default way)



```
library Abc net/hep/bla/bla/bla/*.java
path_append CLASSPATH "${THISPACKROOT}/classes/Abc.jar"
alias Abc "java net.hep.bla.bla.bla.main"
```

- defined in standard fragments java, jar
- CMT has full control

CMT & Java

(simple way)



```
path_append CLASSPATH "${THISPACKROOT}/classes/Abc.jar"  
private  
make_fragment ThisPack  
document ThisPack ThisPack1 GNUmakefile
```

```
${CONSTITUENT} :: ${THISPSACKROOT}/classes/Abc.jar
```

```
${THISPACKROOT}/classes/Abc.jar :
```

```
cd $(src);\n${MAKE} install
```

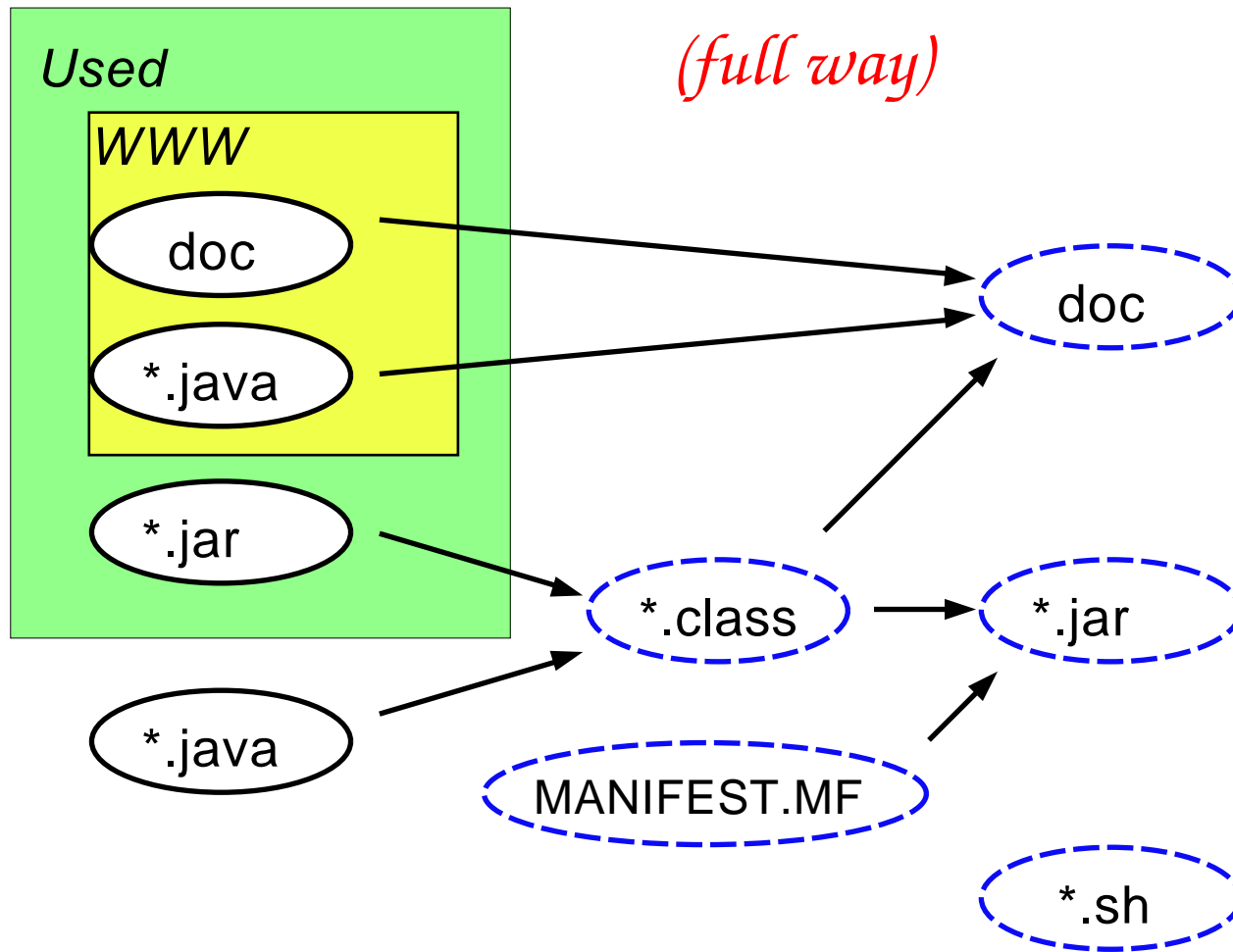
```
${CONSTITUENT}clean ::
```

```
cd $(src);\n${MAKE} clean
```

- CMT used only to path env
- all building standalone
- it's possible as (almost) all dependencies go via CLASSPATH

CMT & Java

(full way)



- full control by CMT
- not yet implemented

CMT & Java

(issues)



- JavaDoc:
 - CMT should have accesses to JavaDoc pages of used packages (to be able to create links), with proper versions
 - standard options (and top-files) should be created if don't exist
- Ant:
 - de-facto standard for Java configuration
 - CMT-requirements <-> Ant-config conversions would be usefull
- JAR:
 - proper MANIFEST.MF should be created
 - dependencies should be handled
 - executable JAR should be created when applicable
- Build reasults can be stored in CVS as they are platform/compiler independent (to large extend) => not necessary to always build
- Dependencies between packages mediated mostly just by proper CLASSPATH => simple configuration
- Advanced features:
 - bytecode engineering (run-time -> *.java, *.class) / enhancement (*.class -> *.class)
 - JNI/Invocation (dependencies between *.h, *.c(xx), *.java, *.class)