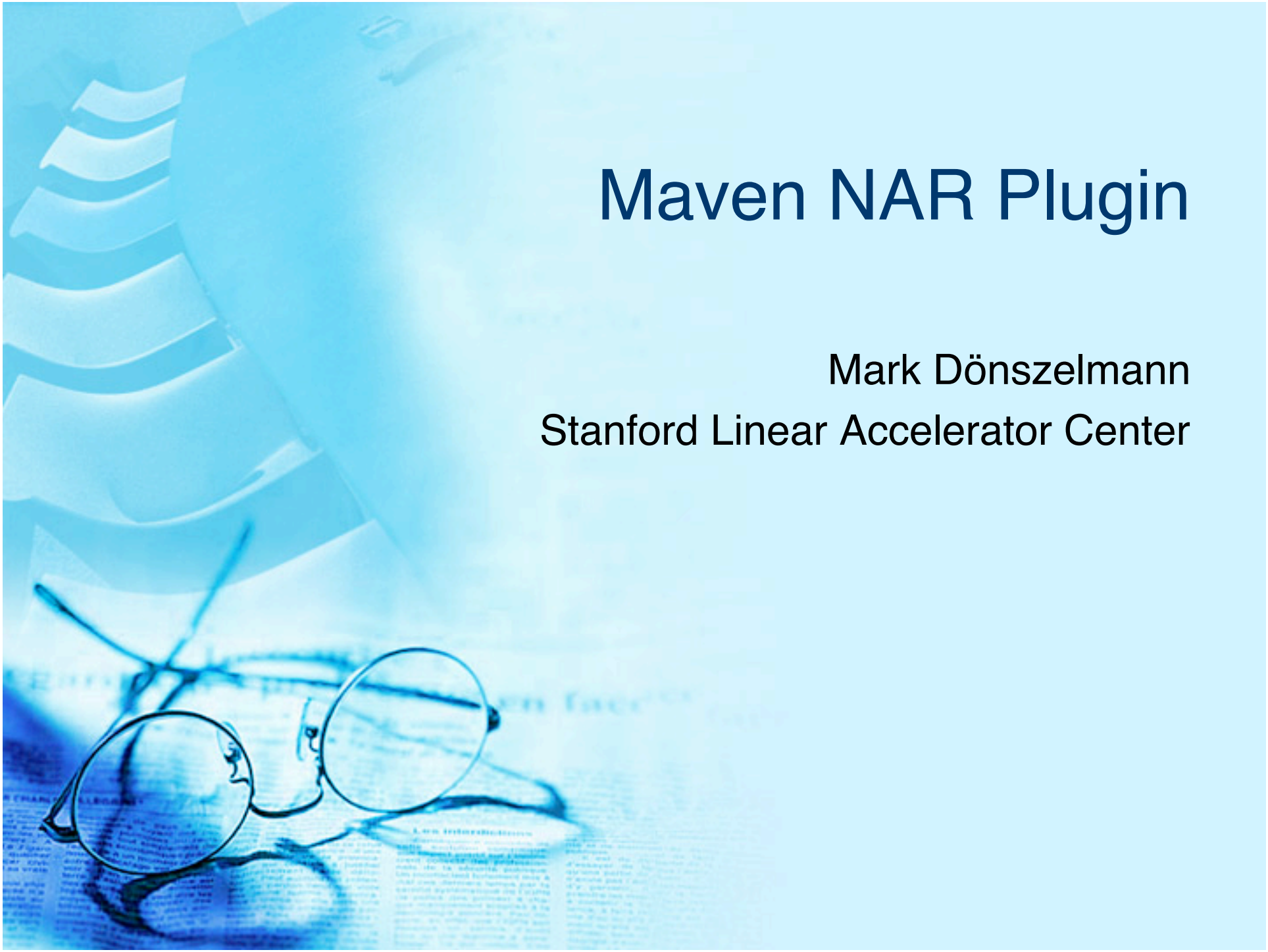


Maven NAR Plugin

Mark Dönszelmann
Stanford Linear Accelerator Center

Content

- Why we created the NAR Plugin
- What is a Native Archive (NAR) ?
- What is Architecture-OperatingSystem-Linker (AOL) ?
- How does it all work ?
- NAR Plugin
  - Usage
  - Goals
- NAR and Maven Philosophy
- Problems
- Ideas

Why a Native Plugin for Maven

- Migrate our Java (and C++/Fortran) code to maven.
- Looked at the maven-native-plugin, 1 year ago, (both maven 1 and maven 2) and found:
  - + Very configurable
  - - Did not run out of the box (no defaults)
  - - No binary dependencies
  - - Not cross platform (different profiles for different platforms)
- So: we wrote our own, the NAR (Native Archive) Plugin

Native Archive Plugin

- For Maven 1 and 2
- Compiles and links native code
- On Linux, Windows, MacOS X, Solaris, …
- With gcc, g++, Microsoft CL, CC, …
- Creates Native Archives for deployment
- Allows dependencies on Native Archives
- Uses standard maven mechanisms (deploy, download)
- Fully configurable, but works out of the “box”
- One configuration for multiple platforms

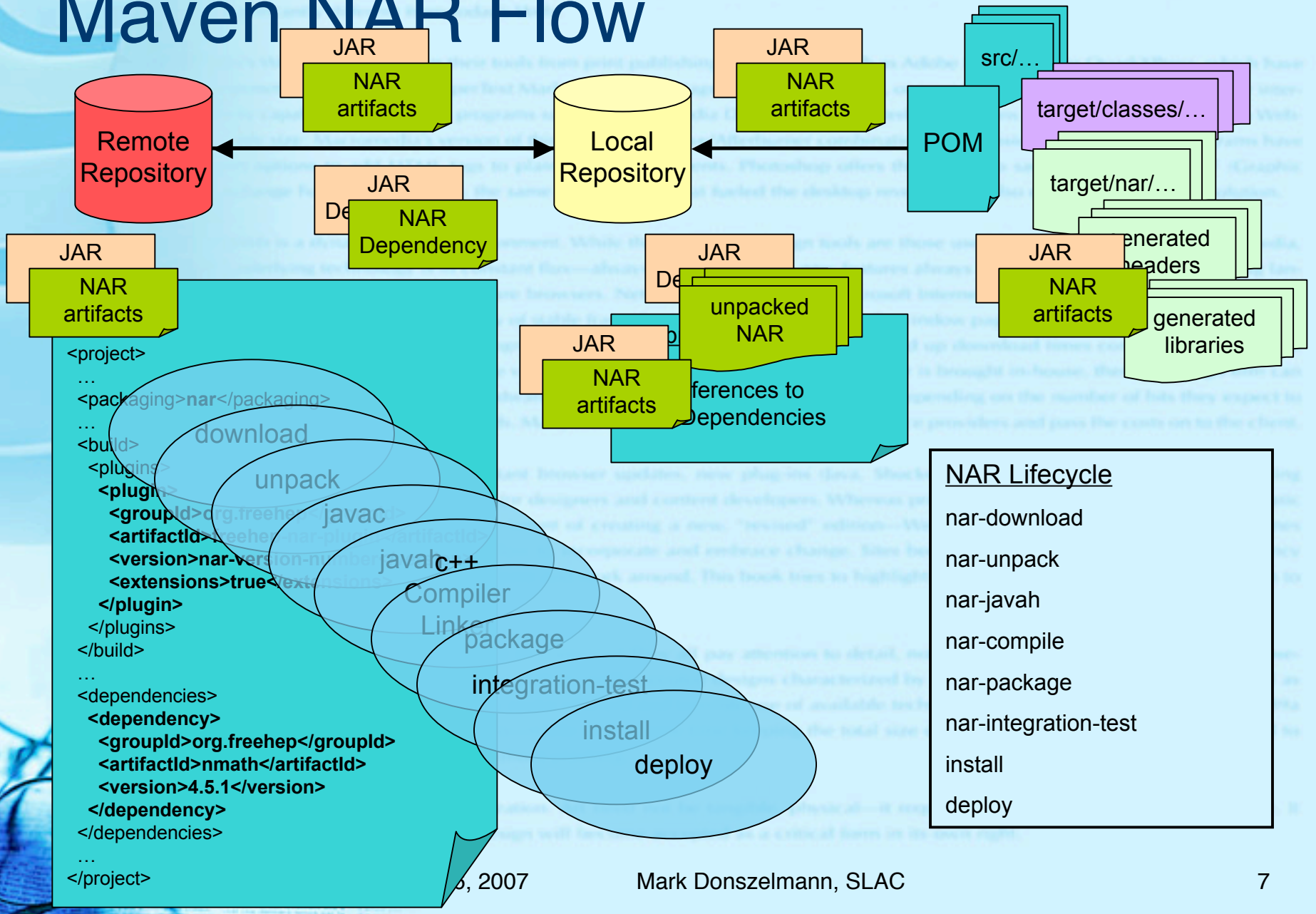
Native Archive

- nar is a jar file (with ".nar" extension) and consists of a collection of native files:
  - Header files
  - Libraries
  - Binaries
  - Data files
- A nar is an attached artifact to a jar artifact.
  For example for a native math library:
  - nmath-4.7.jar - java archive if applicable
  - nmath-4.7-noarch.nar - machine independent archive (headers)
  - nmath-4.7-<AOL>-<type>.nar - machine dependent archive (one or more)
- nar files can be
  - uploaded and published on maven servers (just like jar files)
  - depended on (just like jar files)

AOL Classifier

- AOL Classifier specifies where the nar file was created and where it will work:
  - Architecture
    - i386, x86, amd64, ppc, sparc, …
  - Operating System
    - Windows, Linux, MacOSX, SunOS, …
  - Linker
    - g++, gcc, msvc, CC, icc, icpc
- Examples:
  - x86-Windows-msvc, x86-Windows-g++
  - i386-Linux-g++, i386-Linux-icpc, amd64-Linux-g++
  - ppc-MacOSX-g++, i386-MacOSX-g++
  - sparc-SunOS-CC

Maven NAR Flow



NAR Lifecycle

- nar-download
- nar-unpack
- nar-javah
- nar-compile
- nar-package
- nar-integration-test
- install
- deploy

Maven NAR Plugin Goals

- nar-download
  - downloads nar dependencies
- nar-unpack
  - unpacks all nar dependencies
- nar-javah
  - creates header files for classes with native methods
- nar-compile and nar-testCompile
  - compiles all native (test) code
- nar-test
  - runs native test *currently inactive*
- nar-package
  - bundles up nar files
- nar-integration-test
  - runs integration test against native libraries
- nar-assembly
  - assembles distributions of combinations of nar files (multi-platform)

- Goals
  - can be configured and run separately
  - are configured in the “nar” lifecycle

Usage: shared library

```
<project>
  ...
  <packaging>nar</packaging>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>org.freehep</groupId>
        <artifactId>freehep-nar-plugin</artifactId>
        <version>nar-version-number</version>
        <extensions>true</extensions>
      </plugin>
    </plugins>
  </build>
</project>
```

- Creates a shared library
- Default compiler and linker settings
- Note the “nar” packaging
- Note the <extensions> tag

Usage: JNI Library

```
<project>
  ...
  <packaging>nar</packaging>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>org.freehep</groupId>
        <artifactId>freehep-nar-plugin</artifactId>
        <extensions>true</extensions>
        <configuration>
          <libraries>
            <library>
              <type>jni</type>
            </library>
          </libraries>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

- Creates a JNI library
- Note the configuration section

Configuration

```
...
    <plugin>
      <groupId>org.freehep</groupId>
      <artifactId>freehep-nar-plugin</artifactId>
      <configuration>
        <java>
          <include>true</include>
        </java>
        <javah>
          <excludes>
            <exclude>**/ModuleJNI.class</exclude>
          </excludes>
        </javah>
        <libraries>
          <library>
            <type>jni</type>
          </library>
        </libraries>
      </configuration>
    </plugin>
```

- Configuration
  - add java include files
  - run javah but exclude one specific class
  - create a JNI library

NAR Lifecycle

Phase (standard jar lifecycle phases)	*Goals (NAR Goals in **bold**)*
generate-sources	**nar-download**
process-sources	**nar-unpack**
process-resources	resources
compile	compile, **nar-javah**
process-classes	**nar-compile**
process-test-resources	testResources
test-compile	testCompile, **nar-testCompile**
test	test, **nar-testCompile**
package	**nar-package**, jar
integration-test	**nar-integration-test**
install	install
deploy	deploy

initialization

- Derive and set some properties for usage in other goals.
- A(rchitecture) and O(perating System) of AOL are deduced from the machine we run on, but can be overridden.
- AOL.properties file defines rest of the defaults.
- L(inker) of AOL is specified in a AO dependent property:
  - ppc.MacOSX.linker = g++
- All other settings are AOL dependent properties:
  - ppc.MacOSX.g++.c.compiler = gcc
  - ppc.MacOSX.g++.fortran.options = -Wall -fno-automatic …

nar-download

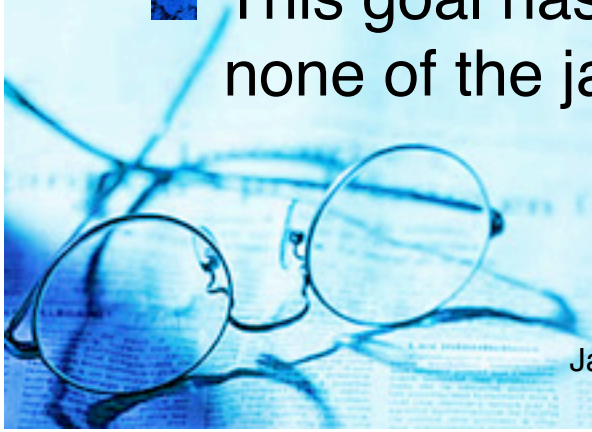
- Assumes maven has downloaded any declared dependencies of type “jar”
- Looks into these jar dependencies for the file
  - META-INF/nar/***groupId***/***artifactId***/nar.properties
  - containing a list of nar files to download:
    - groupId/artifactId-noarch.nar
    - groupId/artifactId-${aol}-static.nar
    - groupId/artifactId-${aol}-shared.nar
- All nar files are downloaded from a remote maven repository and stored in the local maven repository (unless they were already downloaded before).

nar-unpack

- All refered nar files are of no direct use to native compilers.
- They are unpacked (unless this has happened already before) into the local repository.
  Nar files are never unpacked on the remote repository.
- Compilers and Linkers can now refer to libraries (.so, .a, .dll), header files (.hh, .h) and other configuration files.
- Tools can also refer to binary executables that were packed in nar files,

nar-javah

- Runs javah (just after java class compilation) on any class with “native” methods to produce header files.
- Scans .class files for native methods, not .java files.
- The javah tool is picked up from the java installation used by maven.
- It runs with a CLASSPATH specified by the configuration, the project class files and all “jar” dependencies.
- You can also set a BOOTCLASSPATH.
- This goal has no effect if there are no java sources or none of the java classes have native methods.



nar-compile and nar-testCompile

- Compiles and Archives/Links any native (test) code under:
  - src/main (normally in src/main/c++, src/main/fortran, ..)
  - src/test …
- Uses CPPTasks to handle unified compiler and linker options and general handling of compiler/archiver/linker.
- Will invoke correct compiler based on extension of the source files.
- Compiler Include PATHS:
  - any directories specified in the configuration
  - the path to the header files produced by nar-javah
  - the path to the header files of any unpacked nar dependencies
- Linker Search PATHS:
  - any directories specified in the configuration
  - the path to libraries of any unpacked nar dependencies
  - the path to the java virtual machine library (if applicable)

nar-package

- Creates the nar files (-noarch and -aol-type)
  - nmath-4.5.1-noarch.nar
  - nmath-4.5.1-ppc-MacOSX-g++-jni.nar
- Creates the nar.properties file
  - with references to the above files
  - places it in the right place for pickup
- Standard package (jar) goal
  - picks up any resource files, including the nar.properties files
  - picks up any java class files
  - creates nmath-4.5.1.jar

nar-integration-test

- Runs tests against
  - the packaged jar file
  - the created libraries
  - any dependent jar files
  - any dependent (and unpacked) nar files
- Copy of the standard test goal
  - turn on forking by default
  - setup java.library.path to load shared libraries
- Currently only useful to test JNI libraries

install and deploy

- Install (default jar goal)
  - copies the jar file and its attached nar files to the local repository.
  - unpacking of the nar is done by a project that depends on it in the nar-unpack goal.
- deploy (default jar goal)
  - copies the jar file and its attached nar files to a remote repository.
  - nar files are **never** unpacked in the remote repository.

nar-assembly

- For each listed AOL do:
  - Download all dependencies
  - Unpack all dependencies
  - Copy all library related files to target
- Then the standard assembly plugin will:
  - copy other necessary files to target
  - bundle target into .zip and .tar.gz, …

```
…
<plugin>
  <groupId>org.freehep</groupId>
  <artifactId>freehep-nar-plugin</artifactId>
  <configuration>
    <classifiers>
      <classifier>x86-Windows-msvc</classifier>
      <classifier>ppc-MacOSX-g++</classifier>
      <classifier>i386-Linux-g++</classifier>
    </classifiers>
  </configuration>
  <executions>
    <execution>
      <goals>
        <goal>nar-download</goal>
        <goal>nar-unpack</goal>
        <goal>nar-assembly</goal>
      </goals>
    </execution>
  </executions>
</plugin>
…
```

Maven 2 Philosophy

- From the Maven Book:
  - Convention over Configuration
    - Standard Directory Layout
    - Single Project produces Single Output
    - Standard Naming Conventions
  - Reuse of Build Logic
  - Declarative Execution
  - Coherent Organization of Dependencies

Convention over Configuration

- Standard Directory Layout for Native Projects
  - Parallel to java

```
/yourproject
        /src
            /main
                /java
                /resources
                /include
                /c++
                /c
                /fortran
            /test
                /java
                /resources
                /include
                /c++
                /c
                /fortran
```

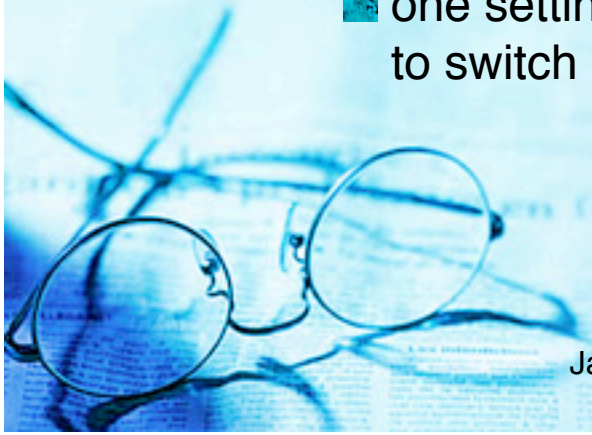
Convention over Configuration

- A single Native Project produces a single Output
  - The NAR artifacts are **attached artifacts** to the primary output
  - Why?
    - Native libraries, object files and headers make a complete set. AOL specific nar files have the same functionality, just for different machines. Between -noarch and all -aol-type nar files none seems to be an obvious primary artifact.
    - A JNI library consists of both java classes in a jar file and some shareable library. The jar would logically depend on the shareable library, however javah needs the classes in the jar file to generate the header file, needed for compilation of the shareable library. No way to make this work, except by attaching the shareable library as an artifact to the jar file.
    - Dependencies are declared on jar artifacts, without specifying Architecture, OS and Linker. The POM stays generic and cross-platform. Since most nar files are AOL specific, they would need to be attached.

Convention over Configuration

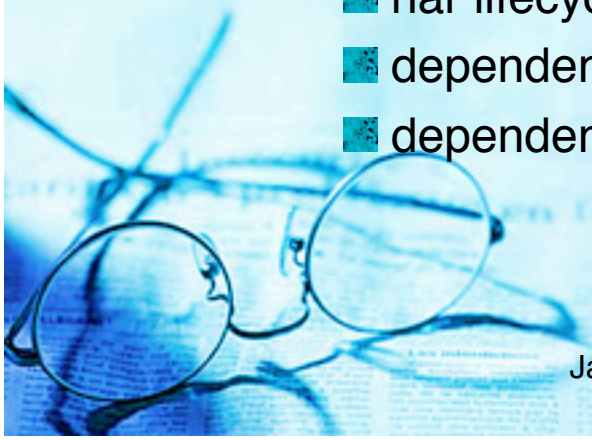
- Standard Naming Conventions
  - Output as attached artifacts
    - artifactId-version-noarch.jar
    - artifactId-version-aol-type.nar
  - Unified interface (CPPTasks) to Native Compilers and Tools
    - for java compilers and tools this is the case
    - for Native tools CPPTasks handles the job
    - one setting in the configuration to switch on debugging

```
...
<plugin>
  <groupId>org.freehep</groupId>
  <artifactId>freehep-nar-plugin</artifactId>
  <configuration>
    <cpp>
      <exceptions>false</exceptions>
      <debug>true</debug>
    </cpp>
  </configuration>
</plugin>
...
```



Reuse and Execution

- Reuse of Build Logic
  - nar lifecycle integrates nicely with the jar lifecycle
  - separate configuration of nar goals is possible
  - most goals executed via the NarManager which provides programmatic API for other plugins (outside the NAR Plugin) which needs to deal with nar artifacts.
- Declarative Execution
  - nar integrates best into maven using the nar lifecycle
  - standard pom is used to declare
    - nar lifecycle
    - dependency on NAR Plugin
    - dependencies on artifacts that have attached nar artifacts



Organization of Dependencies

- Remote Repository

```
remoterepo/
    org/
        freehep/
            nmath/
                4.5.1/
                    nmath-4.5.1.pom
                    nmath-4.5.1.pom.sha1
                    nmath-4.5.1.jar
                    nmath-4.5.1.jar.sha1
                    nmath-4.5.1-noarch.nar
                    nmath-4.5.1-noarch.nar.sha1
                    nmath-4.5.1-MacOSX-g++-static.nar
                    nmath-4.5.1-MacOSX-g++-static.nar.sha1
                    nmath-4.5.1-MacOSX-g++-shared.nar
                    nmath-4.5.1-MacOSX-g++-shared.nar.sha1
                    nmath-4.5.1-Linux-g++-shared.nar
                    nmath-4.5.1-Linux-g++-shared.nar.sha1
                    nmath-4.5.1-Windows-msvc-shared.nar
                    nmath-4.5.1-Windows-msvc-shared.nar.sha1
                    ...
```

- Local Repository

```
repository/
    org/
        freehep/
            nmath/
                4.5.1/
                    nmath-4.5.1.pom
                    nmath-4.5.1.pom.sha1
                    nmath-4.5.1.jar
                    nmath-4.5.1.jar.sha1
                    nmath-4.5.1-noarch.nar
                    nmath-4.5.1-noarch.nar.sha1
                    nmath-4.5.1-MacOSX-g++-shared.nar
                    nmath-4.5.1-MacOSX-g++-shared.nar.sha1
                    ...
                    nar/
                        bin/
                            ppc-MacOSX-g++/
                                NMath
                        include/
                            nmath/
                                NMath.hh
                        data/
                            NMath.data
                        lib/
                            ppc-MacOSX-g++/
                                shared/
                                    libNMath.so
                                    libNMathExtra.so
```

Availability

- Version
  - 2.0-alpha-5
- Currently in our FreeHEP library
  - http://java.freehep.org/freehep-nar-plugin
- FreeHEP Maven 2 server
  - http://java.freehep.org/maven2

Unsupported

- Creation of binaries
- nar-test goal (should maybe use cppunit?)
- nar-integration-test goal only works for JNI libraries
- Linking with shared libraries
- Choice of linking type
- Creation of -noarch nar without creating -aol nar.
- Preservation of disk-space by deleting nar files that have been unpacked.
- And more…

Problems

- CPPTasks
  - Seems not very actively maintained. We made some changes. Does make a unified approach to compilers/linkers and their options.
- Configuration
  - Seemed easier in maven 1 with its property files.
- SNAPSHOTS
  - Publishing attached artifacts as SNAPSHOT seems to create sequence numbers that are not in synch with the primary output (jar file).
  - SNAPSHOTS of attached artifacts are not re-downloaded
- AOL
  - Not specific enough. Maybe we need a map to handle libs that are backwards compatible.

Ideas

- Standard Configure/Make (GNU) packages could be wrapped by some Maven Plugin into nar files.
- Move the NAR Plugin (at some point) to CodeHaus or Apache.
- Add Doxygen to NAR (a la javadoc).