Using Maven to build OpenCV

This page describes the how to build OpenCV using Apache Maven. The Maven build is simply a wrapper around the existing CMake process but has the additional aims of creating Java OSGi-compatible bundles with included native support and also allow the build to be carried out on RaspberryPi (ARM) architecture. There is nothing preventing using the POM on x86 Linux however.

The following assumes building on Debian-based Linux platform.

1 - Overview

The Maven build process aims to:

1. Provide a simpler route to build OpenCV and Java bundles.
2. Automatically check the required native dependencies.
3. Make the Java libraries OSGi compatible.
4. Include the native OpenCV native library inside the Java bundle.
5. Integration testing of the bundle within an OSGi environment.
6. Allow the build to function on x86, x86_64 or amd architectures, Debian-based Linux platform.

2 - Preparing The Build environment

To build using the Maven build process both `Maven` and and up-to-date `JDK` (Java Development Kit) need to be installed. If you know you already have these installed then continue to `Environment Variable` otherwise the easiest solution is to install them using the aptitude package manager:

```
sudo aptitude install maven default-jdk
```

Note that installing via `aptitude` you are unlikely to get the latest version of Maven or JDK although if you are not developing Java code this shouldn't matter for this build process.

3 - Starting the build

3.1 - Environment variables

Applicability: All processors.

The following environment variables must be set otherwise the build will fail and halt:

- `$JAVA_HOME` (the absolute path to the JDK root directory)
- `$ANT_HOME` (the absolute path to the Ant root directory)

It is recommended that advantage is taken of multiple processor cores to reduce build time. This can be done by setting a MAKEFLAGS environment variable specifying the number of parallel builds e.g.:

- `$MAKEFLAGS="-j8"`

However if this flag is not set the build will NOT fail. On a RaspberryPi 2 typical build times are 5 hours with `-j1` (which is the default if `$MAKEFLAGS` is not specified) and a little over 2 hours with `-j4`.

All of the above environment variables can be set on an ad-hoc basis using 'export'.

3.2 - Build Directory

Applicability: All processors

By default the following build directories are created.

`<OpenCV_root_dir>/build`

`<OpenCV_root_dir>/build/maven/opencv/target`

`<OpenCV_root_dir>/build/maven/opencv-it/target`

Under `build` are the standard OpenCV artifacts. Under `build/maven/opencv/target` can be found the OSGi compatible Java bundle. When deploying the bundle into an OSGi framework e.g. Apache Karaf, loading of the native library is automatically taken care of. An integration testing module is created under the `opencv-it` directory and is only of use during the build but is disabled by fault. The standard Java library as created by the CMake process is also available as specified in the existing OpenCV documentation.

The Maven build is initiated from the directory contain the `pom.xml` file.

3.3 - x86 or x86_64 Architecture:

Generally all that is required is the standard Maven command:

```
mvn clean install
```

One of the first things the build will do is check the required native dependencies. The Maven build indicates the status of the required dependencies and will fail at this point if any are missing. Install using the package manager e.g. aptitude or apt-get, and restart the build with the above command.

Once the build successfully completes the OSGi compatible artifacts are available as described above in 'Build Directory'.

3.4 - ARM 32-bit Architecture - Raspbian Distribution

Similar to the x86 architecture the native dependencies are first checked so install any that are missing, however at the time of writing there are no official `libtbb2` and `libtbb-dev` packages in Raspbian. Version 4.4.3 of Intel's Thread Building Blocks library are available here as a Raspbian-compatible Debian packages.

PLEASE NOTE THESE ARE NOT OFFICIAL RASPBIAN PACKAGES. INSTALL AT YOUR OWN RISK.

The build can be started with the following command:

```
mvn clean install
```

Upon a successful build the libraries will be available as described above in 'Build Directory'.

3.5 CMake

Applicability: x86 processors

The CMake Maven plugin is configured to use the native CMake package (recommended) i.e. it will NOT download the latest CMake binary. Should you require CMake download then include the following Maven commandline switch when building:

```
-Ddownload.cmake=true
```

3.6 Integration Tests

Applicability: All processors

OSGi integration tests can be run as part of the build by including the following commandline switch to Maven:

`-Pintegration`

4.0 Maintainer Notes

This section is relevant to those maintaining the Maven platform build. If you just want to build the library then you do not need to refer to this section.

4.1 Updating POM Version to Match Core Version

Maven requires the version to be hard-coded in the POM or in otherwords it cannot be changed at runtime. When the core C/C++ code version changes it is easy to forget to update the Maven version. The POM utilises the enforcer plugin to ensure the POM and Core versions match causing the build to fail if they do not.

Should the POM version require updating then this can be done utilising the Maven 'versions' plugin and this will apply the correct version to all POMs within the project. Execute the following Maven command from the root directory of the Maven project:

`mvn versions:set -DnewVersion=$(. ./opencv/scripts/functions && cd ./opencv/scripts && extract_version && echo $REPLY)`