```
package org.apache.maven.shared.osgi;

/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.  See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */

import java.io.File;
import java.io.IOException;
import java.util.Enumeration;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.jar.JarFile;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.zip.ZipEntry;

import org.apache.maven.artifact.Artifact;

import aQute.lib.osgi.Analyzer;

/**
 * Default implementation of {@link Maven2OsgiConverter}
 *
 * @plexus.component
 *
 * @author <a href="mailto:carlos@apache.org">Carlos Sanchez</a>
 * @version $Id$
 */
public class DefaultMaven2OsgiConverter
    implements Maven2OsgiConverter
{

    /** Bundle-Version must match this pattern */
    private static final Pattern OSGI_VERSION_PATTERN = Pattern
        .compile( "[0-9]+\\.[0-9]+\\.[0-9]+(\\.[0-9A-Za-z_-]+)?" );

    /** pattern used to change - to . */
    // private static final Pattern P_VERSION = Pattern.compile("([0-9]+(\\.[0-9])*)-(.*)");
    /** pattern that matches strings that contain only numbers */
    private static final Pattern ONLY_NUMBERS = Pattern.compile( "[0-9]+" );

    private static final Pattern DATED_SNAPSHOT =
        Pattern.compile( "([0-9])(\\.([0-9]))?(\\.([0-9]))?\\-([0-9]{8}\\.[0-9]{6}\\-[0-
9]*)" );

    private static final Pattern DOTS_IN_QUALIFIER =
        Pattern.compile( "([0-9])(\\.[0-9])?\\.([0-9A-Za-z_-]+)\\.([0-9A-Za-z_-]+)" );
```

```
    private static final Pattern NEED_TO_FILL_ZEROS = Pattern.compile( "([0-9])(\\.([0-9]))?
(\\.([0-9A-Za-z_-]+))?" );

    private static final String FILE_SEPARATOR = System.getProperty( "file.separator" );

    private String getBundleSymbolicName( String groupId, String artifactId )
    {
        return groupId + "." + artifactId;
    }

    /**
     * Get the symbolic name as groupId + "." + artifactId, with the following exceptions
     * <ul>
     * <li>if artifact.getFile is not null and the jar contains a OSGi Manifest with
     * Bundle-SymbolicName property then that value is returned</li>
     * <li>if groupId has only one section (no dots) and artifact.getFile is not null then
the
     * first package name with classes is returned. eg. commons-logging:commons-logging ->
     * org.apache.commons.logging</li>
     * <li>if artifactId is equal to last section of groupId then groupId is returned. eg.
     * org.apache.maven:maven -> org.apache.maven</li>
     * <li>if artifactId starts with last section of groupId that portion is removed. eg.
     * org.apache.maven:maven-core -> org.apache.maven.core</li>
     * </ul>
     */
    public String getBundleSymbolicName( Artifact artifact )
    {
        if ( ( artifact.getFile() != null ) && artifact.getFile().exists() )
        {
            Analyzer analyzer = new Analyzer();

            try
            {
                JarFile jar = new JarFile( artifact.getFile(), false );

                if ( jar.getManifest() != null )
                {
                    String symbolicNameAttribute = jar.getManifest().getMainAttributes()
                        .getValue( Analyzer.BUNDLE_SYMBOLICNAME );
                    Map bundleSymbolicNameHeader = analyzer.parseHeader(
symbolicNameAttribute );

                    Iterator it = bundleSymbolicNameHeader.keySet().iterator();
                    if ( it.hasNext() )
                    {
                        return (String) it.next();
                    }
                }
            }
            catch ( IOException e )
            {
                throw new ManifestReadingException( "Error reading manifest in jar "
                    + artifact.getFile().getAbsolutePath(), e );
            }
        }

        int i = artifact.getGroupId().lastIndexOf( '.' );
        if ( ( i < 0 ) && ( artifact.getFile() != null ) && artifact.getFile().exists() )
        {
            String groupIdFromPackage = getGroupIdFromPackage( artifact.getFile() );
            if ( groupIdFromPackage != null )
            {
                return groupIdFromPackage;
            }
```

```
        }
        String lastSection = artifact.getGroupId().substring( ++i );
        if ( artifact.getArtifactId().equals( lastSection ) )
        {
            return artifact.getGroupId();
        }
        if ( artifact.getArtifactId().startsWith( lastSection ) )
        {
            String artifactId = artifact.getArtifactId().substring( lastSection.length() );
            if ( Character.isLetterOrDigit( artifactId.charAt( 0 ) ) )
            {
                return getBundleSymbolicName( artifact.getGroupId(), artifactId );
            }
            else
            {
                return getBundleSymbolicName( artifact.getGroupId(), artifactId.substring( 1
) );
            }
        }
        return getBundleSymbolicName( artifact.getGroupId(), artifact.getArtifactId() );
    }

    private String getGroupIdFromPackage( File artifactFile )
    {
        try
        {
            /* get package names from jar */
            Set packageNames = new HashSet();
            JarFile jar = new JarFile( artifactFile, false );
            Enumeration entries = jar.entries();
            while ( entries.hasMoreElements() )
            {
                ZipEntry entry = (ZipEntry) entries.nextElement();
                if ( entry.getName().endsWith( ".class" ) )
                {
                    File f = new File( entry.getName() );
                    String packageName = f.getParent();
                    if ( packageName != null )
                    {
                        packageNames.add( packageName );
                    }
                }
            }

            /* find the top package */
            String[] groupIdSections = null;
            for ( Iterator it = packageNames.iterator(); it.hasNext(); )
            {
                String packageName = (String) it.next();

                String[] packageNameSections = packageName.split( "\\" + FILE_SEPARATOR );
                if ( groupIdSections == null )
                {
                    /* first candidate */
                    groupIdSections = packageNameSections;
                }
                else
                // if ( packageNameSections.length < groupIdSections.length )
                {
                    /*
                     * find the common portion of current package and previous selected
groupId
                     */
                    int i;
                    for ( i = 0; ( i < packageNameSections.length ) && ( i <
```

```
groupIdSections.length ); i++ )
                    {
                        if ( !packageNameSections[i].equals( groupIdSections[i] ) )
                        {
                            break;
                        }
                    }
                    groupIdSections = new String[i];
                    System.arraycopy( packageNameSections, 0, groupIdSections, 0, i );
                }
            }

            if ( ( groupIdSections == null ) || ( groupIdSections.length == 0 ) )
            {
                return null;
            }

            /* only one section as id doesn't seem enough, so ignore it */
            if ( groupIdSections.length == 1 )
            {
                return null;
            }

            StringBuffer sb = new StringBuffer();
            for ( int i = 0; i < groupIdSections.length; i++ )
            {
                sb.append( groupIdSections[i] );
                if ( i < groupIdSections.length - 1 )
                {
                    sb.append( '.' );
                }
            }
            return sb.toString();
        }
        catch ( IOException e )
        {
            /* we took all the precautions to avoid this */
            throw new RuntimeException( e );
        }
    }

    public String getBundleFileName( Artifact artifact )
    {
        return getBundleSymbolicName( artifact ) + "_" + getVersion( artifact.getVersion() )
+ ".jar";
    }

    public String getVersion( Artifact artifact )
    {
        return getVersion( artifact.getVersion() );
    }

    public String getVersion( String version )
    {
        String osgiVersion;

        // Matcher m = P_VERSION.matcher(version);
        // if (m.matches()) {
        // osgiVersion = m.group(1) + "." + m.group(3);
        // }

        /* TODO need a regexp guru here */

        Matcher m;
```

```
/* if it's already OSGi compliant don't touch it */
m = OSGI_VERSION_PATTERN.matcher( version );
if ( m.matches() )
{
    return version;
}

osgiVersion = version;

/* check for dated snapshot versions with only major or major and minor */
m = DATED_SNAPSHOT.matcher( osgiVersion );
if ( m.matches() )
{
    String major = m.group( 1 );
    String minor = ( m.group( 3 ) != null ) ? m.group( 3 ) : "0";
    String service = ( m.group( 5 ) != null ) ? m.group( 5 ) : "0";
    String qualifier = m.group( 6 ).replaceAll( "-", "_" ).replaceAll( "\\.", "_" );
    osgiVersion = major + "." + minor + "." + service + "." + qualifier;
}

/* else transform first - to . and others to _ */
osgiVersion = osgiVersion.replaceFirst( "-", "\\." );
osgiVersion = osgiVersion.replaceAll( "-", "_" );
m = OSGI_VERSION_PATTERN.matcher( osgiVersion );
if ( m.matches() )
{
    return osgiVersion;
}

/* remove dots in the middle of the qualifier */
m = DOTS_IN_QUALIFIER.matcher( osgiVersion );
if ( m.matches() )
{
    String s1 = m.group( 1 );
    String s2 = m.group( 2 );
    String s3 = m.group( 3 );
    String s4 = m.group( 4 );

    Matcher qualifierMatcher = ONLY_NUMBERS.matcher( s3 );
    /*
     * if last portion before dot is only numbers then it's not in the middle of the
     * qualifier
     */
    if ( !qualifierMatcher.matches() )
    {
        osgiVersion = s1 + s2 + "." + s3 + "_" + s4;
    }
}

/* convert
 * 1.string   -> 1.0.0.string
 * 1.2.string -> 1.2.0.string
 * 1          -> 1.0.0
 * 1.1        -> 1.1.0
 */
m = NEED_TO_FILL_ZEROS.matcher( osgiVersion );
if ( m.matches() )
{
    String major = m.group( 1 );
    String minor = m.group( 3 );
    String service = null;
    String qualifier = m.group( 5 );

    /* if there's no qualifier just fill with 0s */
    if ( qualifier == null )
```

```
            {
                osgiVersion = getVersion( major, minor, service, qualifier );
            }
            else
            {
                /* if last portion is only numbers then it's not a qualifier */
                Matcher qualifierMatcher = ONLY_NUMBERS.matcher( qualifier );
                if ( qualifierMatcher.matches() )
                {
                    if ( minor == null )
                    {
                        minor = qualifier;
                    }
                    else
                    {
                        service = qualifier;
                    }
                    osgiVersion = getVersion( major, minor, service, null );
                }
                else
                {
                    osgiVersion = getVersion( major, minor, service, qualifier );
                }
            }
        }

        m = OSGI_VERSION_PATTERN.matcher( osgiVersion );
        /* if still its not OSGi version then add everything as qualifier */
        if ( !m.matches() )
        {
            String major = "0";
            String minor = "0";
            String service = "0";
            String qualifier = osgiVersion.replaceAll( "\\.", "_" );
            osgiVersion = major + "." + minor + "." + service + "." + qualifier;
        }

        return osgiVersion;
    }

    private String getVersion( String major, String minor, String service, String qualifier
)
    {
        StringBuffer sb = new StringBuffer();
        sb.append( major != null ? major : "0" );
        sb.append( '.' );
        sb.append( minor != null ? minor : "0" );
        sb.append( '.' );
        sb.append( service != null ? service : "0" );
        if ( qualifier != null )
        {
            sb.append( '.' );
            sb.append( qualifier );
        }
        return sb.toString();
    }
}
```