How to create a Deb package to install a War file into Tomcat:

Introduction:

First of all here's an explanation of what we are going to do:

Our aim is to create a .deb package to install a war file into the tomcat webapps directory.

The war file will be created using **ant** and the source code of the program. Then this war file will be included into the deb file. We need the source code of the package and the build.xml necessary to create the war file.

In our example, we are going to replace the configuration file for the pmacct daemon. We will also execute a mysql script to create a database and a user. The execution of the mysql script requires the mysql root password. We will use debconf to obtain this info from the user.

We will use debconf libraries in our package to simplify the upgrade process. To avoid user input the information needed will only be asked for during the first installation.

Other considerations:

The example that we use in this How-to is the creation of a package for the software P**mgraph** version **1.0** .

In our example, when the .war file is created it is stored into a directory called “dist” in our “working directory” (see below). The default configuration files for the program will be placed there by Ant. These files will be:

- pmgraph.war
- pmacctd.conf
- pmacct-create-db_v6.mysql

Process

You will need to make sure the following software is installed:

1. javahelper
2. ant
3. build-essential
4. dh-make
5. debhelper
6. devscripts
7. fakeroot

1.- Create a working directory:

Create a directory with the name of the program and the version

eg. for the package pmgraph version 1.0

mkdir pmgraph-1.0

Put all the necessary files needed to create the war file, including the ***build.xml*** that Ant needs, into this directory. (The creation of the War file using Ant is out of the scope of this package).

For the pmgraph project, this means a directory containing the following directories that can be downloaded from the repository. Copy the following files from your source-code (eg from your workspace) into your "*pmgraph-1.0*" directory:

```
build.xml
config
debian
lib
tests
web
```

2.- Debianize the directory:

To create a debian package we need a directory called *debian* with some specific files. ***jh_makepkg*** is a tool which creates the default *debian* directory. To create a package for a jar, we will have to change a few things but this tool allows us to create the directory structure with the default files. From the "working directory" *pmgraph-1.0* execute the command:

```
jh_makepkg
```

This asks you some questions about the package that you want to create:

Package Type :

```
What type of package is it? Application, or Library?
Select:
    [A] Application (Default)
    [L] Library
[Al] $ A
```

Build System:

```
What type of build system does it have? Ant, Makefiles, or None?
Select:
    [A] Ant
    [M] Makefiles
    [V] Maven
    [N] None---make one for me (Default)
[Namv] $ A
```

in our case Ant

and the Java version:

```
Which Java runtime does it need? Free runtime, or Sun?
Select:
    [F] Free compiler/runtime (Default)
```

```
        [5] Sun Java 1.5 (Package must be in contrib)
        [6] Sun Java 1.6 (Package must be in contrib)
        [I] Iced Tea
    [F56i] $ 6
```

3.- The debian directory has now been created

It should contain the following files which need to be edited to customise them for our package:

changelog
compat
control
copyright
pmgraph.install
pmgraph.links
pmgraph.manifest
rules

Follow this link for further information on what each file should contain http://ubuntuforums.org/showthread.php?t=51003 .

Below there is a description of each file, its default content and what it should contain for our example:

- **changelog:** changes from the last package created.

  Default info:

  ```
  package (version) unstable; urgency=low

    * Initial release. (Closes: #XXXXXX)

   -- Noe Gonzalez <noeg@fen-ndiyo3>  Mon, 27 Apr 2009 16:22:44 +0100
  ```

  Example of customized info for the project pmgraph version 1.0:

  ```
  pmgraph (1.0) hardy; urgency=low

    * Initial release. (Closes: #XXXXXX)

   -- Aptivate <info+pmgraph@aptivate.org>  Mon, 20 Apr 2009 12:20:04 +0100
  ```

- **control:** This file contains all the info for the debian package: dependencies, description of the package, requisites to create the deb file etc.

  Default info

  ```
  Source: pmgraph
  Section: contrib/misc
  ```

```
Priority: optional
Maintainer: Noe Gonzalez <noeg@fen-ndiyo3>
Build-Depends: debhelper (>> 5), javahelper , ant
Build-Depends-Indep: sun-java6-jdk
Standards-Version: 3.7.3
Homepage: <homepage>

Package: pmgraph
Architecture: all
Depends: ${java:Depends}, ${misc:Depends}
Description: Short Description
 Long Description
```

Example of customized info for the project pmgraph version 1.0. The red lines show the entries that you need to change.

```
Source: pmgraph
Section: net
Priority: optional
Maintainer: Aptivate <info+pmgraph@aptivate.org>
Build-Depends: debhelper (>> 5), javahelper , ant (>= 1.7)
Build-Depends-Indep: sun-java6-jdk
Standards-Version: 3.7.3
Homepage: http://pmgraph.sourceforge.net

Pre-Depends: debconf
Package: pmgraph
Architecture: all
Depends: tomcat5.5 (>= 5.5), pmacct (>= 0.11), mysql-server
Description: pmGraph is a software application for network monitoring to
work with pmacct
     The purpose of pmGraph is to work with pmacct, a network monitoring
and auditing tool pmacct runs on a firewall, router or bridge, or collects
information from multiple routers, and stores the information in a database.
The database permits powerful analysis, but is difficult to use. pmGraph
helps by providing a simple graphical overview of the network data recorded
by pmacct.
```

Very important lines of this file are:

Depends: dependencies that the package needs to work.
Description: description showed on an apt search.

- **copyright:** license of the software contained in the package (Be careful editing this file it has a specific layout and the creation of the package can fail if the spacing is incorrect). If a error is made the dpkg command returns you a verbose that will make it easy to identify the error line.

  For our example (GNU license) :

  ```
  This package was Debianised by Aptivate <info+pmgraph@aptivate.org> on
  Mon Apr 20 12:20:03 BST 2009
  ```

This package was downloaded from http://pmgraph.sourceforge.net.

Upstream Author: Aptivate <info+pmgraph@aptivate.org>.

This software is copyright (c) 2006-09 by Aptivate, a not-for-profit company registered in England and Wales, company number 04980791.

You are free to distribute this software under the terms of the GNU General Public License.

On Debian systems, the complete text of the GNU General Public License can be found in the file `/usr/share/common-licenses/GPL'.

- **pmgraph.install** : This file contains the list of files that should be copied when the package is installed, that means when the .deb file is installed.

  For our example we are going to copy 3 files into "usr/share/pmgraph". These files are the pmgraph.war, a configuration file for pmacct daemon and a mysql script that will create the mysql database and user.

  The content of the file should be:

  ```
  dist/pmgraph.war usr/share/pmgraph
  dist/pmacct-create-db_v6.mysql usr/share/pmgraph/conf
  dist/pmacctd.conf usr/share/pmgraph/conf
  ```

- **pmgraph.links :** Symbolic link that should be created when the package is installed.

  That is useful if you want to create a link for example from /usr/bin to any file you have copied during the installation like the files in "*usr/share/pmgraph*". In our case that is not useful at all.

  Remove this info to leave an empty file.

- **pmgraph.manifest :** The lines in this file will be added to the manifest file of the jar or war file created with ant. Taking into account that we are going to create a WAR file and that ant already creates the property Manifest file the **pmgraph.manifest** file must be empty. By default this file contains the version of Java and the main class of the application.

  Remove this info to leave an empty file.

- **rules:** The rules file is the script that will be executed to create our .deb package. In this file there are a series of sections each one with a comment saying which kind of command you should place there. There are three sections that we need to change to create a .deb with a War package:

1. ***export JAVA_HOME=/usr/lib/jvm/java-6-sun***

   Modify this line to point your java home dir. This line is especially important if you need to use a specific version of Java. In our case it is necessary to use sun-java-6 and therefore we force to use this version of Java.

Eg if your java path ($ whereis java) starts /usr/bin/java use “/usr”

2. build-indep-stamp:

```
build-indep-stamp:
        dh_testdir
        # Build the package
        ant war   # this line has been added to create the war file
        touch $@
```

3. binary-indep: build-indep install-indep

```
binary-indep: build-indep install-indep
      # Create the package here
      dh_testdir
      dh_testroot
      dh_clean -k
      dh_install -i       # this line copies the files indicated in  packagename.install
      dh_installdocs -i
      dh_installchangelogs -i
      jh_manifest -i      # this line should be removed when building a war file
      dh_link -i
      jh_exec -i
      jh_depends -i -j sun6
      dh_compress -i
      dh_fixperms -i
      dh_installdeb -i
      dh_gencontrol -i
      dh_md5sums -i
      dh_builddder
```

5.- The Postinstall and PostRemove scripts:

At this point we have all we need to create a .deb file but all our .deb file will do when it is installed is copy the files specified in **pmgraph.install**. We need to create a script **packageName.postinst** to do the following additional tasks:

1. Copy the war file to the tomcat webapps dir.
2. Execute the mysql script to create the database.
3. Copy the default pmacct configuration file in the pmacct configuration directory. “/etc/pmacct/”
4. Set the parameter “*TOMCAT5_SECURITY=no*” in the default configuration file of tomcat which is “/etc/default/tomcat5.5”.
5. Restart Tomcat Server and pmacct server so the changes made to the configurations file take effect.

These tasks can be done using a postinstall script. This script, called “packageName.postinst”, should be placed in the debian subdirectory of our working directory (eg pmgraph-1.0/debian). It will be added to the .deb file and will be executed after the installation of the package.

Analogously you can create a script called packageName.postrm which is executed any time after the package is removed.

This scripts are shell script and it is out of the scope of this tutorial to explain how to create a shell script. Info about shell script programming can be found at:

http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html

Here is an example of the script we use to find where the tomcat webapps directory is, to create a link to where the war file of pmgraph is located and to execute the mysql script which creates the database. This script overrides the default pmacct configuration file.

```
#!/bin/bash
CATALINA_DEFAULT_BASE=`grep -B1 CATALINA_BASE /etc/default/tomcat5.5 | grep Default |
sed -e 's/.*: //'`
. /etc/default/tomcat5.5

CATALINA_BASE=${CATALINA_BASE:-$CATALINA_DEFAULT_BASE}
ln -sf /usr/share/pmgraph/pmgraph.war $CATALINA_BASE/webapps/pmgraph.war

echo Please, enter Mysql root passwd
stty_orig=`stty -g`          # avoid the pass been printed in the console
stty -echo
read PASSWD
stty $stty_orig

# Create database and user.
echo Creating Mysql data base...
mysql -u root -p$PASSWD < /usr/share/pmgraph/conf/pmacct-create-db_v6.mysql


# Copy default pmacct config file
echo Configuring pmacct to work with mysql database and use pmacct user...
test -s /etc/pmacct/pmacctd.conf && mv /etc/pmacct/pmacctd.conf
/etc/pmacct/pmacctd.conf.old
cp /usr/share/pmgraph/conf/pmacctd.conf /etc/pmacct/pmacctd.conf

echo do you want to set tomcat security off (y/n) ?
read RET

if [ "$RET" = "y" ]; then
       JAVASECURITY=`grep -B1 TOMCAT5_SECURITY /etc/default/tomcat5.5 | grep -v \# |
sed -e 's/.*=//'`
       if [ "$JAVASECURITY" != "no" ]; then
              echo "TOMCAT5_SECURITY=no" >> /etc/default/tomcat5.5
       fi
fi

# restart tomcat and pmacct server
if [ -x "/etc/init.d/tomcat5.5" ]; then
        if [ -x "`which invoke-rc.d 2>/dev/null`" ]; then
                invoke-rc.d tomcat5.5 restart || exit $?
        else
                /etc/init.d/tomcat5.5 start || exit $?
        fi
fi

if [ -x "/etc/init.d/pmacct" ]; then
        if [ -x "`which invoke-rc.d 2>/dev/null`" ]; then
                invoke-rc.d pmacct start || exit $?
        else
                /etc/init.d/pmacct start || exit $?
        fi
fi

db_stop        # in case invoke fails
exit 0
```

6.- Create the package:

Now we have created our scripts and modified the files in the *"work directory/debian"* we can create our deb file by simply going to our "working directory" "~/pmgraph-1.0" and executing the command:

```
dpkg-buildpackage -rfakeroot
```

This command creates 4 files in the parent directory of our "working directory"

Directory list for the pmgraph example:

- **pmgraph_1.0_all.deb**
- pmgraph_1.0.dsc
- pmgraph_1.0_i386.changes
- pmgraph_1.0.tar.gz

Now using *dpkg -i pmgraph_1.0_all.deb* you can install your package.

7.- Using debconf in our postinstall script:

At the moment our post install script is requesting some info from the user that means that if the package is going to be updated, the user will be prompted again for this information and the installation update will be stopped till the user responds. To avoid that we are going to use debconf to request info from the user. The info obtained from the user the first time the software is installed will be kept in a database and when the software is updated the data will be obtained from this database without needing to prompt the user again.

To make our postinstall script work with debconf, we will have to make a few changes in our "working directory"/debian/ files. First of all, we need to create a Template File. The template file will contain all the info messages and questions that the user is prompted for.

The name of the template file will be packageName.templates. For our example this is:

*debian/**pmgraph.templates***

The contents of this file for our example are:

```
Template: pmgraph/mysql
Type: password
Default: secret
Description: Enter Mysql server root password.
 Mysql root password to create the pmacct database and the pmacct user.

Template: pmgraph/java_security
Type: boolean
Default: true
Description: Disable Java security?
 To avoid putting a lot of exceptions in java security properties file it is better to
```

disable java security

Once we have our templates file we need a config file called packageName.config For our example this is:

*debian/**pmgraph.config***

This config file is a script which will be executed before the package installation by the package manager, and which will collect all the information necessary form the user and keep it in the debconfig database. After the installation the post script will be executed obtaining the data from the database which means that it will not need to prompt the user. Subsequent installations or updates will obtain data from the debconf database and will not prompt the user.

Detailed information about how to create a template and config file can be obtained at http://www.fifi.org/doc/debconf-doc/tutorial.html.

An example of config script that uses the Template file detailed above is:

```
#!/bin/sh -e

# Source debconf library.
. /usr/share/debconf/confmodule

# Do you like debian?
db_input high pmgraph/mysql || true
db_go
```

In this case the script is a shell script but it is possible to use perl scripts. If you are using shell script you need to add the line ". /usr/share/debconf/confmodule " which loads the file with all the functions necessary to use debconf. Basically what the other commands do is:

db_input: show a message to the user asking for a data or just showing info depending on the type of template used.

db_go: execute all the db_input messages issued before.

Then this script simply asks the user a question (the text of the question is the defined in the pmgraph.templates file which is " *Enter Mysql server root password.* ") and the user's response will be kept then in the pmgraph.postinstall.

Remove these lines from the pmgraph.postins script

```
echo Please, enter Mysql root passwd
stty_orig=`stty -g`          # avoid the pass been printed in the console
stty -echo
read PASSWD
stty $stty_orig
```

And replace them with

```
db_get pmgraph/mysql
PASSWD=$RET
```

The $RET variable contains the value of the last command executed. The command *"db_get pmgraph/mysql"* means make a query to the database for "pmgraph/mysql ".

Remove these lines from the pmgraph.postins script

```
echo do you want to set tomcat security off (y/n) ?
read RET
if [ "$RET" = "y" ]; then
```

And replace them with

```
db_get pmgraph/java_security
if [ "$RET" = "true" ]; then
```

7 B.- Using debconf in our postinstall script:

To include the **pmgraph.config, pmgraph.template** files in our deb file we have to change the /debian/rules file and include the command ***dh_installdebconf.*** *See the line in red below.*

```
#!/usr/bin/make -f
#
# Sample debian/rules that uses javahelper.
# This file was generated by jh_makepkg and may be used
# without restriction. It was inspired by the dh-make
# sample debian/rules

# Uncomment this to turn on verbose mode.
#export DH_VERBOSE=1

export JAVA_HOME=/usr

# Put depended upon jars in here
# export CLASSPATH=

build: build-arch-stamp build-indep-stamp
build-arch: build-arch-stamp
build-arch-stamp:
	dh_testdir
	touch $@

build-indep: build-indep-stamp
build-indep-stamp:
	dh_testdir
	# Build the package
	ant war
	touch $@

clean:
	dh_testdir
```

```
        dh_testroot
        #ant clean
        dh_clean
        rm -f build-stamp

install-indep: build-indep
        dh_testdir
        dh_testroot
        dh_clean -k
        dh_installdirs

        #ant install

binary-arch: build-arch
        # Java packages are arch: all, nothing to do here

binary-indep: build-indep install-indep
        # Create the package here
        dh_testdir
        dh_testroot
        dh_clean -k
        dh_install -i
        dh_installdocs -i
        dh_installchangelogs -i
        dh_link -i
        jh_exec -i
        jh_depends -i -j sun5
        dh_compress -i
        dh_fixperms -i
        dh_installdebconf  # add deb conf scripts to DEBIAN dir
        dh_installdeb -i
        dh_gencontrol -i
        dh_md5sums -i
        dh_builddeb -i

binary: binary-indep binary-arch
.PHONY: build build-arch build-indep clean binary-indep binary-arch binary install-indep
```

Once we have added this file to the debian directory, we need to execute the command *dpkg-buildpackage -rfakeroot* again to recreate the deb file with the postinstall scripts being implemented. Remember that this command must be executed from the "working directory".

8.- Apt repositories how they build the package:

Reached this point we have a debian package which we can use to install pmgraph, however if we want tu upload the package to an APT repository, usually is not the debian package what must be uploaded to the repository. Most of the APT repositories usually build the package by them selves therefore what we must provide to it is all what it needs to create the debian file. To do so we can execute the following command being in the work directory:

```
dpkg-buildpackage -S
```

This command create a ".dsc" file which is the file that the APT repository will use to create the debian package, it create a few other files that contain info about changes in the package and the source code. For our example the files created are:

```
pmgraph_1.0.dsc
pmgraph_1.0_source.changes
pmgraph_1.0.tar.gz
```

Sometimes the build in the Apt repository fails because the environment in which the package is created in the repository could be different from the local one. We can create a local environment to test the ".dsc" files before uploading it to the repository. To create a local clean environment we will use pbuilder detailed info about how to use it could be found in:

https://wiki.ubuntu.com/PackagingGuide/Complete#Packaging%20With%20CDBS

apt-get install pbuilder.

But the basic commands are:

- to create a pbuilder environment, run

```
sudo pbuilder create --distribution <ubuntu_version> \
      --othermirror "deb http://archive.ubuntu.com/ubuntu  <ubuntu_version>
main restricted universe multiverse"
```

- to build a package using `pbuilder`, run

```
sudo pbuilder build *.dsc
```

- to update a pbuilder environment, run

```
sudo pbuilder update
```

In our specific case we are using java-sun-6 and to include it in the building environment we will need to accept the java sun licence, to do so it is necessary to create a few files

1. Create in our home directory a file called **.pbuilderrc** containing:

```
called from $0"

 # where to go when things go wrong
        HOOKDIR=~/.pbuilder-hooks
```

2. In the home directory create a directory called **pbuilder-hooks** and into it a file called **D50sun-java-licences** with execution privileges and the following content:

```
#!/bin/sh
debconf-set-selections <<EOF
sun-java5-jdk shared/accepted-sun-dlj-v1-1 boolean true
sun-java6-jdk shared/accepted-sun-dlj-v1-1 boolean true
EOF
```

Now we can execute the command *sudo pbuilder build *.dsc* to emulate what the apt repository do when we upload the ".dsc" file to it.