ATACAMA LARGE MILIMITER ARRAY

ALMA COMMON SOFTWARE

ACS UTFSM TEAM

Packaging and Distribution
–
RPM Generation Procedure

Doc. No. USM-DIS-0002

Issue 1.2

Date 26/01/2005

Prepared for Review

Keywords: ACS, UTFSM, RPMS, Packaging

Prepared . . . . . Mauricio Araya 29/12/2004
Name Date Signature

Approved . . . . . Rodrigo Araya . . . . .
Name Date Signature

Released . . . . . Mauricio Araya . . . . .
Name Date Signature

This page was intentionally left blank

Change Record

Issue/Rev.	Date	Section/Parag. affected	Reason/Initiation/Documents/Remarks
1.0	29/12/04	All	First Draft.
1.1	29/12/04	All	Spell check and grammar corrections.
1.2	25/01/04	5	Added section ACS UTSFM RPMs.

This page was intentionally left blank

Contents

This page was intentionally left blank

1 INTRODUCTION

1.1 PURPOSE

This document has all the information harvested by the ACS-UTFSM TEAM during the realization of the "Zero Approach" project, and consist in all the information, procedures and techniques used to package ACS software using RPMs.

1.2 SCOPE

The attendance public of this document are ACS packagers, sysadmins and any person that want to install ACS through RPMs technology.

1.3 REFERENCE DOCUMENTS

[1] LSO-INS-ESO-00500-0001 – Guide for Document Preparation
[2] USM-GUI-0001 – Document Preparation Guidelines
[3] USM-DIS-0001 – Project 001 Summary
[4] http://www.rpm.org/max-rpm/ – Maximum RPM

1.4 ABBREVIATIONS AND ACRONYMS

ACS	ALMA Common Software
CMM	Configuration Management Module
ESO	EUROPEAN SOUTHERN OBSERVATORY
GNU	GNU is Not Unix
RPM	RPM Package Manager
SRPM	Source RPM
VLT	Very Large Telescope

1.5 GLOSSARY

1.6 STYLISTIC CONVENTIONS

The following styles are used:

bold
in the text, for commands, filenames, pre/suffixes as they have to be typed.

italic
in the text, for parts that have to be substituted with the real content before typing.

`teletype`
for examples

<name>
in examples, for parts that have to be substituted with the real content before typing.

bold and *italic* are also used to highlight words.

2 OVERVIEW

Every Unix based system has a *package management system*, that helps installing, uninstalling, verifying, querying, and updating computer software packages. RPM is the most known package manager in the Linux world, because very important vendors like Red Hat inc. uses RPM to manage all the software in their platforms. Almost all non-redhat based distributions support RPMs as well, due that Open Source Projects usually have at least one RPM version.

2.1 ACS PACKAGING SNAPSHOT

ACS is distributed in two ways.

The first one provides precompiled binaries and files packaged in a tarball. The idea is to *decompress & run* ACS in few steps, following a simple "cookbook" installation manual. Throughout this document this type of installation will be call *Full-Binary* distribution.

The second, provides only the source code of ACS that should be compiled using the *make* tool and shell scripts. Unlike the *Full-Binary* distribution, this *Source-Code* distribution is a little more flexible in terms of which part of ACS we want to install. Also, compiling ACS will avoid any architecture dependency. The main problem of this installation is that compiling ACS takes time.

The source code of ACS is divided in modules with the CMM format (VLT legacy), and each one provides a stand alone compilation Makefile. In the logical sense each module can be thinked as a package, but some modules has a strong dependency with other modules, so is very difficult to install them separately.

2.2 RPM INTRODUCTION

RPM is a command line driven package management system that stores all the information of software packages installed using RPM. A binary database contains each file associated to the package, making easier the installation, removal, querying and updating. The command line to use RPM database is `rpm(8)` , and all usage information can be found in the official manual (`man rpm`). For a detailed information about the usage, please refer to [4]

3 GENERAL RPM PACKAGE MANAGEMENT

A RPM package is a binary file (compressed) that contains multiple type files, including binary executable programs, scripts, configuration files, libraries, etc. Also a RPM package contains meta-information on where and how to install, remove and update those files. RPM cares about dependencies: if a software package depends on other, the software cannot be installed till all the dependencies are installed[1].

There are four usual options that represents the whole life cycle of a RPM package:

- Query: Check the existence of a RPM at the internal database. Example:

[1] You can force the installation, but is not recommended in any case.

```
[root@marge root]# rpm -q rpm
rpm-4.3.1-0.3

[root@marge root]# rpm -ql rpm
/bin/rpm
/etc/cron.daily /rpm
/etc/logrotate. d/r pm
/etc/rpm
/usr/bin/gendif f
/usr/bin/rpm2cp io
/usr/bin/rpmdb
/usr/bin/rpmque ry
/usr/bin/rpmsig n
/usr/bin/rpmver ify
/usr/lib/librpm -4. 3. so
.
.
.
```

- Install: Install a RPM package into the system. You must be sure that the software was not installed before, because RPM will duplicate the installation. Example:

```
# Simple Installation
[root@marge root]# rpm -i acsDevelKit-3.1- 1.i386.rpm
# Verbose Installation
[root@marge root]# rpm -ivh acsDevelKit-3. 1-1.i386. rpm
```

- Update: Use the new RPM package to update the installation. If the software is not installed, this version will be installed. Normally, people use this option to install RPMs, because is safer. Example:

```
# Simple Updating
[root@marge root]# rpm -U acsDevelKit-3.1- 2.i386.rpm
# Verbose Updating
[root@marge root]# rpm -Uvh acsDevelKit-3. 1-2.i386. rpm
```

- Remove: Deletes every file of the package from the system. Example:

```
[root@marge root]# rpm -e acsDevelKit
```

The RPM package filename structure is resumed as follows:

name-A.B-C.arch .rpm

name: The Package/Software name.
A: The Major version of the Software.
B: The Minor version of the Software.
C: The Release of the RPM (version of the RPM).
arch: The hardware architecture for compiled binaries. This can be i386,i686,alpha,sparc,ia64, or noarch. If the package says src then is a source-code package and is not directly installable.

4 RPM GENERATION

An installable RPM must be generated compiling the source code and making a "fake install" into a safer jail (or compile-farm). For this purpose a special command is used called rpmbuild(8) driven by a file that contains the RPM specifications and rules called specfile . Obviously the source code and the related patches are needed too.

4.1 PRECONDITIONS TO MAKE A RPM

- The first step before making a RPM is to be sure that the methods for autocompiling and installing (like make) are well-defined. The ideal is to have a complete division between the compiling and installing because in the building of a RPM those stages are independent. No complete paths at the installation is allowed, because a fake-root is used to make RPMs.
- The software source code must be organized such only one compressed file can contains the whole source code and program files. RPMs only support tarballs (gzip or bzip2).
- A compile-farm is needed as well. In RedHat/Fedora the root user can use the system compile-farm located at /usr/src/redhat . We strongly recommend to make your own compile-farm as a user.

4.2 MAKING A USER COMPILEFARM

A compilefarm is a known directory structure to make possible the automatic make of RPMs. This directory structure includes:

```
— compilefarm/
   |— BUILD
   |— RPMS
   |— SOURCES
   |— SPECS
   |— SRPMS
   '— TMP
```

The BUILD directory will be the place where the software will be compiled but not installed. At the RPMS and SRPMS directories, rpmbuild(8) will store completed RPMs and Source RPMs. The SOURCE directory must previously contain all the tarballs and patches needed to make the RPM/SRPM. Before making any RPM a *specfile* must be created and stored in the SPECS directory. At last, TMP will be the fake-root directory, where the software will be installed using the TMP directory as the / directory.

After creating this directory structure (that can be anywhere, but must have the right permissions), we must configure rpmbuild to support our new compilefarm. The configuration file is located at $HOME/.rpmmacros and the following example explains it:

```
# $HOME/.rpmmacros file
# Include user definitions to common RPM variables.

%packager Your Complete Name <youremail@company.foo>

# The path to the compilefarm and the compilefarm definition.
```

```
%_topdir                /home/user/comp   ilefarm
%_rpmtopdir             %{_topdir}
%_builddir              %{_topdir}/BUIL   D
%_rpmdir                %{_topdir}/RPMS
%_sourcedir             %{_topdir}/SOUR   CES
%_specdir               %{_topdir}/SPEC   S
%_srcrpmdir             %{_topdir}/SRPM   S
%_tmppath               %{_topdir}/TMP

# If there are unpackaged files that was created during the process,
# the RPMBuild must end.
%_unpackaged_fi   les_terminate_build 1
```

4.3 MAKING A SPECFILE

Making the specifications and rules is the central part of making a RPM. A *specfile* is almost like a shell script. In any section you can use shell script commands to make any preparation, installation or modification. The difference between a *specfile* and a shell script is their structure and meaning.

A *specfile* is divided in 6 parts:

4.3.1 Header

The header contains all information that we discuss at the preconditions section.
Example:

```
# acsDevelKit   Specfile
# ------------   ----

# HEADER

# Short description
Summary:        Alma Common Software Development Kit
# Package Name
Name:           acsDevelKit
# Major and Minor number
Version:        3.1
# Rpm Release
Release:        3
License:        LGPL
Group:          ACS
Packager:       ACS-UTFSM Team <acs@listas.inf   .utfsm.cl>
# The source code is by default in the SOURCE directory
Source:         %{name}-%{versi   on}.tar.gz
URL:            http://www.eso   .org/~almamgr/AlmaAcs/
# Packages needed to build this package
BuildRequires:  pdksh
# Packages needed to install the binary RPM
Requires:       pdksh
# fake-root directory name
```

```
BuildRoot:      %{_tmppath}/%{name}-%{version}-root-%(id -u -n)

# Full description to add to the RPM database
%description
Test, no description available.
```

4.3.2 %prep Section

The preparation stage includes all the before compiling arranges. This includes uncompressing the code, applying patches, make some directories, etc. There are two macros that help this stage: %setup and %patch . The first is for uncompressing the %Source target (that it can be a URL) and put it in the right directories with the right permissions. The second one used to apply the patches to the code in a safe way. More information about RPM macros at [4]. Continuing with the example, a %prep section can be like:

```
%prep
%setup -q

# Preparation of a FAKE /alma/ACS-X.Y directory as a compile farm.
# Is not so clean but it is a preparation example.

TOOLS_INSTALL_DIR="$RPM_BUILD_ROOT/alma/%{group}-%{version}"
rm -rf $RPM_BUILD_ROOT
mkdir -p $TOOLS_INSTALL_DIR
cd $TOOLS_INSTALL_DIR

# We assume that External Tools are installed in the following directories
ln -s /alma/%{group}-%{version}/ant $TOOLS_INSTALL_DIR/ant
ln -s /alma/%{group}-%{version}/gnu $TOOLS_INSTALL_DIR/gnu
ln -s /alma/%{group}-%{version}/JacORB $TOOLS_INSTALL_DIR/JacORB
ln -s /alma/%{group}-%{version}/java $TOOLS_INSTALL_DIR/java
ln -s /alma/%{group}-%{version}/mico $TOOLS_INSTALL_DIR/mico
ln -s /alma/%{group}-%{version}/Python $TOOLS_INSTALL_DIR/Python
ln -s /alma/%{group}-%{version}/TAO $TOOLS_INSTALL_DIR/TAO
ln -s /alma/%{group}-%{version}/tcltk $TOOLS_INSTALL_DIR/tcltk
ln -s /alma/%{group}-%{version}/Tomcat $TOOLS_INSTALL_DIR/Tomcat
```

4.3.3 %build Section

This section includes all the compiling stage, and must not be so large. In almost all GNU software this section only includes a ./configure command with certain flags, and a make command. An example will be then:

```
%build

export LD_ASSUME_KERNEL=2.4.18

./configure --with-a-flag=0
make clean
make all
```

4.3.4 %install Section

The idea of this section is to copy only the files that we want to be at the RPMs with from the BUILD directory to the RPMROOT directory. A example could be:

```
%install

make install INSTALL_ROOT="$ RPM_BUILD_ROOT"
```

4.3.5 %files Section

This section includes all the file list that the RPM database should manage. It supports wildcards and macros, so it should not be a major problem. Example:

```
%files
%defattr(-,root ,root,-)
/alma/ACS-%{ver sion}/ACSSW/
```

4.3.6 %changelog Section

This last section is the changelog record. The format is very strict, so be careful. Each entry use two lines, the format of the first one is date packager <mail> . And the second is the comment of changes. date must be at the format DayWeek Month DayMonth Year , using only the 3 initial letters of DayWeek . In any case, the command,

```
$> echo 'LC_ALL="C" date +"%a %b %d %Y"'}
```

could help.

An example of the changelog is:

```
%changelog
* Tue Dec 21 2004 %{packager}
  Adding a line to the changelog.
* Mon Dec 20 2004 ACS-UTFSM Team <acs@listas.inf .utfsm.cl>
  First Test Dummy RPM
```

4.4 GENERATION PROCEDURE

1. Be sure that all the preconditions are fulfilled, this includes making a compile-farm and define your .rpmmacros file.
2. Make a tarball with the software, and name it name-version.tar .gz. You can use tar -czvf name-version.t ar.gz name-version command to do this.
3. Move the tarball to the compile-farm, specifically to the SOURCE directory.
4. Move any patch to the SOURCE directory.
5. Make the *specfile*, or just modified it to satisfy this new version (i.e. change the version, release, changelog). Then move it to the SPECS directory.
6. Run the command

```
rpmbuild -ba compilefarm/SPECS/name.spec
```

This command will build all stages and possibilities of RPMs, including installable RPMs and SRPMs.

7. Publish or distribute those RPMs.

4.5 MAKING RPMS FROM SRPMS

Source RPMs have a big advantage: they are autocompilable packages, that can be ported to any architecture to make the RPM for that machine. There are two ways to make an RPMs from a SRPM:

- One is to install the SRPM (`rpm -Uvh name-X.Y-Z.src.rpm`). Automatically all the parts of the RPMs (*specfile*,tarballs,patches,etc), will be placed at the compilefarm as it corresponds. Then, the RPM can be generated with the procedure described above.
- The second one is to use all the defaults, and run the following command:

```
rpmbuild --rebuild name-X.Y-Z.src.rpm
```

SRPMs is a way to distribute source code as well, because includes not only the source code, but a standard installation procedure at the *specfile*

5 ACS UTFSM RPMs

ACS UTFSM RPMs are build using *rpack* utilities. ACS UTFSM RPMs provide a full ACS installation.

5.1 rpack INSTALLATION

This utilities are located at HQ CVS. They must be downloaded to build the RPMS. To be able to download rpack you need a CVS account.

```
[ntroncos@nephelle ACS-UTFSM]$ export CVSROOT=:pserver:ntroncos@cvssrv.hq.eso.org:/p
[ntroncos@nephelle ACS-UTFSM]$ cvs login
Logging in to :pserver:ntroncos@cvssrv.hq.eso.org:2401/project2/CVS
CVS password:<passwrd>
[ntroncos@nephelle ACS-UTFSM]$ cvs co ACS/REM/rpack
cvs server: Updating ACS/REM/rpack
U ACS/REM/rpack/ChangeLog
cvs server: Updating ACS/REM/rpack/config
U ACS/REM/rpack/config/rpackTarball.conf
cvs server: Updating ACS/REM/rpack/config/specs
U ACS/REM/rpack/config/specs/acsAnt.spec
U ACS/REM/rpack/config/specs/acsBase.spec
U ACS/REM/rpack/config/specs/acsGnu.spec
U ACS/REM/rpack/config/specs/acsJacOrb.spec
U ACS/REM/rpack/config/specs/acsJava.spec
U ACS/REM/rpack/config/specs/acsMico.spec
U ACS/REM/rpack/config/specs/acsOmniOrb.spec
```

```
U ACS/RPM/rpack/config/specs/acsPython.spec
U ACS/RPM/rpack/config/specs/acsTao.spec
U ACS/RPM/rpack/config/specs/acsTcltk.spec
U ACS/RPM/rpack/config/specs/acsTomcat.spec
cvs server: Updating ACS/RPM/rpack/config/templates
U ACS/RPM/rpack/config/templates/rpackTemplate-AcsDevelKit.spec
U ACS/RPM/rpack/config/templates/rpackTemplate-AcsExtProd.spec
U ACS/RPM/rpack/config/templates/rpackTemplate-AcsSW.spec
cvs server: Updating ACS/RPM/rpack/src
U ACS/RPM/rpack/src/Makefile
U ACS/RPM/rpack/src/rpackBuildAll
U ACS/RPM/rpack/src/rpackBuildExpert
U ACS/RPM/rpack/src/rpackBuildFarm
U ACS/RPM/rpack/src/rpackBuildPackage
U ACS/RPM/rpack/src/rpackMakeTarball
```

rpack directory structure:

```
— ACS/
   '— RPM/
       '— rpack/
           |— config/
           |   |— specs/
           |   '— templates/
           '— src/
```

- ACS/RPM/rpack/config/specs/ contains the spec files (discussed previously) for the RPM generation.

- ACS/RPM/rpack/config/templates/ contains spec file templates.

- ACS/RPM/rpack/config contains the file rpackTarball.conf. This file contains the tarfiles available for an RPM generation. File example:

  ```
  [package] acsGnu
  [mkdir] INSTALL
  [mkdir] PRODUCTS
  [cp] ExtProd/INSTALL /buildGNU INSTALL/
  [cp] ExtProd/INSTALL /checkGNU INSTALL/
  [cp] ExtProd/INSTALL /standardPrologue INSTALL/
  [cp] ExtProd/INSTALL /standardEpilogue INSTALL/
  [cp] ExtProd/PRODUCTS/gnu.tar.gz PRODUCTS/
  [/package]
  ```

  [package] <name> directive indicates the package name.
  [mkdir] <directory> directive indicates that <directory> must be created at ACS_ROOT.
  [cp] <source> <destination> directive indicates that a recursive copy must be from <source> to <destination> .

- ACS/RPM/rpack/src/ contains the rpack scripts.

5.2 rpack SCRIPTS AND USAGE

For the proper functioning of the rpack scrips the full ACS sources are required.

5.2.1 rpackBuildAll

This is a blind user tool, it will create and install all RPMS whose spec files are available. This tool *must run with root privileges* and it must be supplied the release argument (-r). This tool will fail if ACS/ACS_VERSION is not present, in that case the version must be supplied by command line. example:

```
[root@nephelle src]# ./rpackBuildAll -r 1
cat: ../../..//ACS_VERSION: No such file or directory

[root@nephelle src]# ./rpackBuildAll -r 1 -v 4.0
build process
```

All possible arguments are the following:

```
[root@nephelle src]# ./rpackBuildAll -h
rpackBuildAll -- Generates all the RPMs .

Usage: rpackBuildAll -r <release> [ -s <srcdir> -v <version> -c <compfarm>] | -h

-r <release> : is the release number.
-s <srcdir> : Base ACS Source directory, by default ../../../
-v <version> : Redefine the Version of the tarball, defined at ACS/ACS_VERSIO N
-c <compfarm> : Compilefarm, by default ../redhat/
-h : Display this help message
```

5.2.2 rpackBuildFarm

The purpose of this tool is to create a RPM compilefarm (discussed previously). This tool may be run as any user, but the user must have write permission at the destination directory. The tool will create the destination directory and within it the compilefarm. Example:

```
[ntroncos@nephelle src]$ ls /big/compilefarm
ls: /big/compilefarm: No such file or directory
ntroncos@nephelle src]$ ./rpackBuildFarm -c /big/compilefarm
[ntroncos@nephelle src]$ ls /big/compilefarm/
BUILD RPMS SOURCES SPECS SRPMS TMP
```

All possible arguments are the following:

```
[ntroncos@nephelle src]$ ./rpackBuildFarm -h
rpackBuildFarm -- Generates a compilefarm at the specified directory.

Usage: rpackBuildFarm -h | -c <DestDir>

-c <DestDir> : Destination directory where the compilefarm will be created.
-h : Display this help message
```

5.2.3 rpackMakeTarball

This tool will create a proper tarball for the RPM creation. The caveat of this tool is that it will only create tarballs for the packages specified in rpackTarball.conf (discussed previously). To obtain a list of possible packages you must supply the -l option. To build the tarball you supply an available package name. The tool will create the default destination directory ../tarballs and the tarballs within. Example:

```
[ntroncos@nephe   lle src]$ ./rpackMakeTar   ball -l
acsGnu
acsTcltk
acsJava
acsTao
acsAnt
acsJacOrb
acsPython
acsOmniOrb
acsMico
acsTomcat
acsBase
[ntroncos@nephe   lle src]$ ./rpackMakeTar   ball -n acsTcltk
acsTcltk-4.0/
acsTcltk-4.0/.b   ash_profile.acs
acsTcltk-4.0/IN   STALL/
acsTcltk-4.0/IN   STALL/buildTcltk
acsTcltk-4.0/IN   STALL/buildCheckFileExist
acsTcltk-4.0/IN   STALL/checkTcltk
acsTcltk-4.0/IN   STALL/standardProlog ue
acsTcltk-4.0/IN   STALL/standardEpilog ue
acsTcltk-4.0/PR   ODUCTS/
acsTcltk-4.0/PR   ODUCTS/tcltk.tar.gz
```

All possible arguments are the following:

```
[ntroncos@nephe   lle src]$ ./rpackMakeTar   ball -h
rpackMakeTarbal   l is an automatic   tarball   builder   for RPM generation,   developed   b

Usage: rpackMakeTarba   ll -n <name>  [ -c <cfgfile>   -s <srcdir>   -b <bashprof>   -v <ve

-n <name>          : is the package  name, a full  list  of names  with the -l option
-c <cfgfile>    : Configuration   file path, by default   ../config/rpackT   arball.conf
-s <srcdir>     : Base ACS Source  directory,   by default   ../../../
-b <bashprof>   : .bash_profile.   acs path, by default   ../../..//LGPL   /acsBUILD/conf i
-v <version>    : Redefine  the Version  of the tarball,  defined  at ACS/ACS_VERSION
-d <dest>       : Destination   directory,   by default   ../tarballs
-l              : List the possible   packages
-h              : Display  this help  message
```

5.3 rpackBuildPackage

This tool will create the RPM packages at the compilefarm. The following preconditions are necessary:

- the compilefarm must be created.
- the tarballs should be at the SOURCE directory of the compilefarm.

To obtain a list of possible packages you must supply the -l option. This list is made up from the available specfiles in ACS/RPM/rpack/ config/specs/. For the package creation you must supply a name(-n package name), a version(-v ACS version), and a release(-r package version), these are the mandatory option. The caveat of this tool is that it **WILL** overide all rpmmacross defined by the user. example:

```
[ntroncos@nephelle src]$ ./rpackBuildPackage -l
acsGhu
acsTcltk
acsJava
acsTao
acsAnt
acsJacOrb
acsPython
acsOmniOrb
acsMico
acsTomcat
acsBase
[ntroncos@nephelle src]$ ./rpackBuildPackage -n acsTcltk -v 4.0 -r 1
/home/ntroncos/ACS-UTFSM/ACS/RPM/rpack/src
Executing(%prep): /bin/sh -e /home/ntroncos/ACS-UTFSM/ACS/RPM/rpack/compilefarm/TMP/r

BUILD PROCESS

- exit 0
```

The newly created RPM is located in compilefarm/RPM/<ARCH>. All possible arguments are the following:

```
[ntroncos@nephelle src]$ ./rpackBuildPackage -h
./rpackBuildPackage is an automatic RPM maker builder for RPM generation, developed

Usage: rpackBuildPackage -n <name> -v <version> -r <release> [ -c <compfarm> ] | -h

-n <name>       : is the package name, a full list of names with the -l option
-v <version>    : Set the version of the package (needed)
-r <release>    : Set the release of the package (needed)
-c <compfarm>   : Set the compilefarm, by default: ../compilefarm
-p <packager>   : Set the packager name, by default RPM autopackager <acs@listas.inf. ut
-l              : List the possible packages
-h              : Display this help message
```