



Atacama Large Millimeter Array

ALMA Software Development Tools and Integration Guidelines

COMP-70.80.00.00-002-L-GEN

Version: L

Status: (*Draft, Pending, Approved, Released, or Obsolete*)

2005-05-06

Prepared By:		
Name(s) and Signature(s)	Organization	Date
Paola Sivera	ESO	2005-05-06
Approved By:		
Name and Signature	Organization	Date
Released By:		
Name and Signature	Organization	Date



ALMA Project
Title
 ALMA Software Development Tools and
 Integration Guidelines

Doc # : COMP-70.80.00.00-002-L-GEN
 Date: 2005-05-06
 Status: Draft
(Draft, Pending, Approved, Released, Superseded, Obsolete)
 Page: 2 of 43

Change Record

Version	Date	Affected Section(s)	Author	Reason/Initiation/Remarks
A	2003-02-15	All	P. Sivera	All First draft
B	2003-02-28	All	P. Sivera	Integrated first comments
C	2003-05-28	All	P. Sivera	2 Reference Documents: added quick reference about CVS; 4.2:added ITS web page; 8:restructured; 10: added tagging system for releases Rn.
D	2003-07-30	All	P. Sivera	Chapter 10: Added par. 10.12: distribution of integrated ALMA SW (after results of CDR1); 10.12: changed deadline for INTLIST from R0 to R1. Some minor fixes here and there in the documents.
E	2003-09-10	All	J. Uphoff	Clarified TAT how-to.
F	2004-01-20	All	P. Sivera	Eccs-tat merge: updated tat chapter; integrated decisions from Santa Fe meeting
G	2004-02-13	All	P. Sivera	Section2: changed ref doc 9 according to the changes in ALMA EDM from S. Oliver
H	2004-05-25	All	L. Cryer	Reformatted
I	2004-05-31	2, 6, 7, 10	P. Sivera	Added information about ACS-ARCHIVE integration, corrected deadlines, definition of when a subsystem release or an integrated release is ready; corrected link and reference. For CDR2
J	2005-02-17	17	M. Pasquato	Updated information about subsystem directory structure in order to make it compliant with new naming conventions.
K	2005-05-02	9.3	M. Pasquato	Updated MODROOT setting information as required by ALMASW2005015
L	2005-05-06	2, 7.4, 10.10, 10.13, 12.1, 12.4	P. Sivera	Updated for CDR3



ALMA Project
Title
ALMA Software Development Tools and
Integration Guidelines

Doc # : COMP-70.80.00.00-002-L-GEN
Date: 2005-05-06
Status: Draft
(Draft, Pending, Approved, Released, Superseded, Obsolete)
Page: 3 of 43

Table of Contents


1	SCOPE AND PURPOSE	5
2	REFERENCE DOCUMENTS	5
3	LIST OF ABBREVIATIONS/ACRONYMS	6
4	GETTING STARTED.....	7
4.1	Contact persons for trouble-shooting, questions and support.....	8
4.2	Useful web pages.....	8
5	ENVIRONMENT.....	9
6	DIRECTORY STRUCTURE.....	10
6.1	MODROOT and definition of a module.....	11
6.2	INTROOT.....	12
7	MAKEFILE AND CODING STANDARDS	12
7.1	System wide make definitions: acsMakefile	13
7.2	Software module Makefile	13
7.3	Subsystem Makefile	13
7.4	Coding standards	14
8	CVS AND HOW THE SUBSYSTEM SOFTWARE SHOULD BE ORGANIZED	15
8.1	The Repository Structure.....	15
8.2	Subsystem Makefile	16
9	TOOLS FOR AUTOMATED TESTING (TAT).....	18
9.1	Overview	18
9.2	Limitations and future developments	19
9.3	How TAT works.....	19
9.4	Cookbook.....	24
9.5	Commands to remember.....	26
9.6	Some additional features	27
9.6.1	Logfiles	27
9.6.2	How to filter the test output	27
9.6.3	Integration of TAT with JUnit, CppUnit and PyUnit	28
10	MONTHLY INTEGRATION AND ALMA SOFTWARE RELEASES.....	28
10.1	Prerequisites	28
10.2	Integration Periodicity.....	28
10.3	Integration and Testing Strategy	29
10.4	Integration procedure	30
10.5	Testing procedures	30
10.6	Tool(s) for Dynamic Analysis.....	31
10.7	Releases.....	31
10.8	CVS tags for the monthly integration	31
10.9	CVS tags for milestone releases Rn	32
10.10	Integration hardware	34
10.11	Reports	34



ALMA Project
Title
ALMA Software Development Tools and
Integration Guidelines

Doc # : COMP-70.80.00.00-002-L-GEN
Date: 2005-05-06
Status: Draft
(Draft, Pending, Approved, Released, Superseded, Obsolete)
Page: 4 of 43

10.12	Distribution of the integrated software.....	35
10.13	Working Areas organization and new directory structures.....	35
11	SOFTWARE PROBLEM REPORT (SPR) SYSTEM	36
11.1	Overview	36
11.2	Login into the system	37
11.3	Submitting an SPR	37
11.4	Querying the system.....	37
11.5	Changing the login password	38
11.6	ALMA SPR System workflow.....	38
11.7	ALMA Software Board.....	39
12	DOCUMENTATION	39
12.1	Comments inside the code.....	39
12.2	Header of each source file to generate man pages	39
12.3	CVS log file.....	40
12.4	Release Notes	40
12.5	Manuals	41
12.5.1	Templates and Document Number	41
12.5.2	Sitescape	41
APPENDIX A.	A CRASH-COURSE IN BASH SHELL	42

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	<p>Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superseded, Obsolete)</i> Page: 5 of 43</p>
---	--	--

1 SCOPE AND PURPOSE

With this document we intend to give an overview of the standards and tools to be adopted and the procedures and methodologies that have to be followed by the developers within the ALMA SW group. These standards and procedures are defined by the Software Engineering (SE) and Integration, Test and Support (ITS) subsystems, with the aim of giving homogeneity to the produced software and make possible the integration of this software into the ALMA SW release.

To achieve this goal, the cooperation from every developer within the group is crucial and it is extremely important that everybody know about the contents of this document!

These are the topics that will be discussed in the following chapters:


- getting started
- environment
- directory structures
- Makefile
- coding standards
- version control
- tools for automated testing
- monthly integration, releases
- software problem report system
- documentation

Purpose of the document is also to collect in a unique place all the information needed for a newcomer to get confident with the way of working within the ALMA project. Nevertheless, this document can also be useful for people who already have little or good experience in working with the ALMA software.

To avoid repetitions, whenever possible, references to other documents or man pages have been written.

2 Reference Documents

- [1] ALMA-70 .20.00.00-001-A-STD ALMA Software and Hardware Standards, M. Zamparelli
(<http://almaedm.tuc.nrao.edu/forums/alma/dispatch.cgi/Standards/docProfile/100004/d20030527130011/No/ASHS.pdf>)


	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	<p>Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superseded, Obsolete)</i> Page: 6 of 43</p>
---	--	--

- [2] ALMA-SW-0013 Rev. 3 ALMA Software Engineering Practices, M. Zamparelli
(<http://alma.nrao.edu/development/computing/docs/joint/draft/ASEP2.pdf>)
- [3] VLT-MAN-ESO-17200-0908 Rev 1.4 [Tools for Automated Testing User Manual](http://www.eso.org/projects/vlt/swdev/wwwdoc/MAR2001/VLT-MAN-ESO-17200-0908/Output/FronCoverNew.html), P. Sivera
(<http://www.eso.org/projects/vlt/swdev/wwwdoc/MAR2001/VLT-MAN-ESO-17200-0908/Output/FronCoverNew.html>)
- [4] ALMA-70.05.00.00-001-J-PLA ALMA Computing Plan for Phase 2, G. Raffi
- [5] ALMA-SW-9999, Rev 1 ALMA Software Document Review Procedure, M. Zamparelli
(<http://alma.nrao.edu/development/computing/docs/joint/draft/ADRP.pdf>)
- [6] ALMA-80.02.00.00-003-F-STD, ALMA Document Standards, S. Oliver
(<http://almaedm.tuc.nrao.edu/forums/alma/dispatch.cgi/documents/docProfile/100597/d20021126194951/No/t100597.htm/>)
- [7] Open Source Development with CVS, Karl Fogel (<http://cvsbook.red-bean.com/>)
- [8] Version Management with CVS, Per Cederqvist et Al.
(<http://www.cvshome.org/docs/manual/>)
- [9] ALMA-SW-NNNN, Rev 3, CMM to CVS transition, M. Zamparelli,
(<http://almaedm.tuc.nrao.edu/forums/alma/dispatch.cgi/CMMtoCVS/showFile/100004/d20030205105244/No/CMM-CVS-TransitionPlan.pdf>)
- [10] COMP-70.80.00.00-001-G-PLA, Integration, Test and Support Plan, P. Sivera
- [11] Maximizing Your Use of CVS, Dan York
(<http://www.lodestar2.com/people/dyork/talks/2001/ols/frames/frames.html>)

3 List of Abbreviations/Acronyms

<http://www.alma.nrao.edu/development/computing/docs/joint/draft/Glossary.htm>

ALMA	Atacama Large Millimeter Array
ACS	ALMA Common Software
ASB	ALMA Software Board
CVS	Concurrent Version System
GUI	Graphical User Interface

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superseded, Obsolete)</i> Page: 7 of 43
---	---	---

HW	HardWare
ICD	Interface Control Document
ITS	Integration, Test and Support
IDR	Internal Design Review
LCU	Local Control Unit
PC	Personal Computer
PDR	Preliminary Design Review
SCCB	Software Configuration Control Board
SE	Software Engineering
SPR	Software Problem Report
SW	Software
TAT	Tools for Automated Testing
WS	Workstation


4 Getting started

The ALMA project is spread out all over the world. Every site has its own policy and system administrators who will help the newcomers to set up their desktops.

Far from thinking of giving a detailed desktop description, nevertheless, we have defined some standards about the HW and SW to be used. Please, read [1] .

Some basic ideas: the development for the ALMA software shall be done on UNIX, Linux Red-Hat being the chosen distribution. The same platform (Linux Red Hat) has been chosen for Real Time Computers. Java development on Windows is also accepted. About the supported languages, we have C++, C, Java, and Python, with concessions to Tcl/Tk for historical reasons (some tools used within the project have been developed in this language).

Be aware that there are mailing lists to ease the communication within the group. The contact person for addition to the relevant email lists is Brian Glendenning (bglenden@nrao.edu). He is also responsible for adding people to the “Who's Who” web page. Have a look at <http://alma.nrao.edu/development/computing/who/index.html/>

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	<p>Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superceded, Obsolete)</i> Page: 8 of 43</p>
---	--	--

and send him a mail with information about you and, eventually, a picture which will be added to the page.

There are also other tools used within the group for communication purposes, like Yahoo Messenger. (Note: the usernames on Yahoo cannot always be compliant with the naming standards on the UNIX workstation). Ask your supervisor whether and how you should define a login at Yahoo Messenger and how to add your colleagues' logins.

You can also have a look at the web page:

<http://www.eso.org/projects/alma/develop/alma-se/reference/EditorsMessangers.htm>

to have an idea about who is using Yahoo Messenger and with which login.

4.1 Contact persons for trouble-shooting, questions and support

For everything concerning software engineering practices, programming standards, Makefile (see chapter 7) CVS (see chapter 8) and SPR system (see chapter 11), please refer to alma-sw-semgr@nrao.edu (the e-mails alma-sw-semgr@eso.org or almasccm@eso.org work also). For everything concerning installation procedures and monthly integration refer to alma-sw-it@nrao.edu.

4.2 Useful web pages

The entry point for many issues concerning the work with the ALMA software can be found at:


<http://www.eso.org/projects/alma/develop/alma-se/>

From this point you can access, among others, the systems snapshots (reports about the nightly integration and test of the ALMA software), and documentation issued from the SE group, useful FAQs and so on.

A page dedicated to the ITS subsystem only and separated from the SE page work has also been prepared: <http://www.eso.org/projects/alma/develop/software/alma-it/>

There you can find the results of the monthly integrations and tests of the ALMA SW and the progressive work to prepare the ALMA SW releases (see later in this document, in particular chapter 10), as well as the documentation related to the Integration and Test activities. It is also possible to download a tar-gzipped file containing the ALMA SW.

The usage of the web-based collaboration platform called Twiki became also very common within the ALMA project. Every ALMA subsystem has got a Twiki site and developers are encouraged to contribute to the different Twiki topics directly editing the web pages.

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	<p>Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superceded, Obsolete)</i> Page: 9 of 43</p>
---	---	--

The ITS Twiki page is at: <http://almasw.hq.eso.org/almasw/bin/view/ITS/WebHome>. Under the link “Public” there are topics where we expect to see contributions from the different ALMA developers.

5 Environment

As stated in,[2] “The software/hardware environment used by ALMA software developers must be as similar and homogeneous as possible. This is crucial to the project’s success all the more if one considers ALMA’s geographical dispersion”. To achieve this goal, the usage of the ALMA Common Software (ACS) is mandatory. Every developer within the ALMA project shall work on a workstation where the ACS software is installed. In this way, he/she will have available not only all the necessary tools to develop but also a set of environment variables already prepared in order to access those tools in the correct way. (Refer to the web page <http://www.eso.org/~gchiozzi/AlmaAcs/index.html> , where you can find everything related to ACS: releases, documentation, FAQs.)

The installation of ACS is the prerequisite and the starting point. When ACS is installed, a file called `.bash_profile.acs` gets installed under the home directory of the user who performed the installation, in a subdirectory called `.acs`. A copy of this directory is also placed under the path `$ACSRROOT/config/.acs`¹.

This file is written in bash style and supposes that the user uses bash as interactive shell. Then, it is enough to source the file before starting any work and the developer will have the environment setup with the necessary environment variables. To source the file, just run the command:


```
. $ACSRROOT/config/.acs/.bash_profile.acs
```

You can also put that command in your `~/.bash_profile` in order to have the ACS environment automatically sourced each time you do a login.

In case you are not confident with the syntax of the bash shell, a crash course is offered in Appendix A.

Note: to go on with the reading of this document and to be able to try the examples you must have installed the ACS release.

¹ See chapter 6 for the meaning of ACSRROOT

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	<p>Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superceded, Obsolete)</i> Page: 10 of 43</p>
---	---	---

6 Directory structure

To ease the work of the developers, a pre-defined set of standard directories can be created using a simple command:

```
$ getTemplate
```

This is an interactive utility which provides templates for many different purposes. In this case we are interested in choosing the option “directoryStructure”.

Then a choice among different types of directory structures is proposed. Basically you will be interested in using two types:

MODROOT (that can be WS type only, LCU type only or both WS and LCU according to the type of code you have to develop). Here you will put the software sources you have to write;

INTROOT (integration area where the developed code get installed after running “make install”) (for make and Makefile description see chapter 7). You can access this area through the environment variable \$INTROOT. This environment variable can be found in the .bash_profile.acs. It is up to the user to define it in a proper way.


Another important area is called **ACSROOT**. The directory structure is basically the same as for the INTROOT (a part from the Source directory) but the meaning of the area is different: the ACSROOT is the repository for the ACS software. There, all the ACS libraries, tools and the acsMakefile (an extension of the GNU Makefile) are installed and available for every user, who can access that area through the environment variable \$ACSROOT. Only the responsible persons for the installation of the ACS software release can create and populate an ACSROOT. The ownership and permission of the files into the ACSROOT should not allow other users different from the installation user to modify that area.

MODROOT and INTROOT are instead totally handled by each developer, who can create and populate them or delete them.

Please, read [2], in particular chapter 6 to know how those areas are used. (Basically they are accessed through environment variables that are taken into account when building the PATH, LD_LIBRARY_PATH, CLASSPATH and so on. The precedence is always: first MODROOT, then INTROOT, then ACSROOT).

There are also man pages which explain how those areas are organized. You can access them running:

```
man acsDirectoryStructure
```

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	<p>Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superceded, Obsolete)</i> Page: 11 of 43</p>
---	---	---

6.1 MODROOT and definition of a module

The usual location for a MODROOT is somewhere in the user home directory. Each MODROOT corresponds to a software module containing a set of standard subdirectories created with `getTemplate`.

A software module is a piece of software able to perform functions and having an interface available to an external user to access the functions provided.

Technically a module is a way to organize functions in homogeneous groups. The interface hides the implementation and system dependencies from the user.

Managerially the module is the basic unit for planning, project control and configuration control.

There is no rule to define how big a module shall be. Common sense and programming experience should be enough to identify what can be gathered and treated as a unique item. Examples of modules are: a driver for a specific board (the driver itself, install utility, configuration data files, etc.), the logging system of ACS (libraries, utilities, etc.), the configuration database (the server in Java and C++ libraries to communicate with the server)

Once a MODROOT has been created with `getTemplate` you will work under the different directories of the MODROOT, keeping or removing what is not needed. Be aware that some directories are mandatory:

src is the directory where you will be putting the sources you are working on

include is the directory where you will be putting the “.h” files

lib is where the libraries get installed

bin is where the binaries get installed


idl is for the Interface Definition language files²

man is for the man pages

object is where the dependencies files get installed (when running “`make all`”)

doc is for the documentation generated with doxygen (see chapter 12)

² All the external interfaces are now grouped under the ICD subsystem, but the subdirectory “idl” is left in case of need of defining internal interfaces, even for test purposes.

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superceded, Obsolete)</i> Page: 12 of 43
---	--	--

`test` is where the test source code has to be put

6.2 INTROOT

The usual location for the INTROOT is `/introot/<username>`, but any other area accessible to the users on a machine will be OK. This path should be passed to the program “`getTemplate`”, after choosing the options “`directoryStructure`” -> “`createINTROOTarea`”.

Then the variable \$INTROOT must be made available to the user environment. As we saw in chapter 5, you should write in one of your startup files, for example `$HOME/.bash_profile` or `$HOME/.acs/.bash_profile.acs` if you are using bash as interactive shell, the line:

```
export INTROOT=/introot/<username>
```

and log out and in again.

7 Makefile and coding standards

To grant homogeneity in the development of the software for the ALMA project, the GNU make must be used at every site or consortium.

This is by default the case on any system installed with the ACS release according to the ALMA standards.

A wide set of make definitions have been coded and made available through a file called “`acsMakefile`” (see paragraph 7.1).

Analogously, every developer should provide the `src` and `test` directories of his/her software module with a Makefile (see paragraph 7.2) where he/she will indicate the actions to be taken when running “`make all man install`”. This Makefile has to be created starting from the utility “`getTemplate`” and choosing “`code`” and then “`Makefile_for_WS`” (if the code to be produced has to run on a UNIX workstation) or “`Makefile_for_LCU`” (if the code has to be produced using the VxWorks cross compiler). Note that, if the module has been created with the directory structure from `getTemplate`, it will already contain the right Makefile. (The Makefile for LCU code will be soon modified and better defined, because we are initiating the process of migrating to Real time Linux).

The module Makefile must include the `acsMakefile`. (The template contains the adequate instruction).



ALMA Project
Title
ALMA Software Development Tools and
Integration Guidelines

Doc # : COMP-70.80.00.00-002-L-GEN
Date: 2005-05-06
Status: Draft
(Draft, Pending, Approved, Released, Superseded, Obsolete)
Page: 13 of 43

7.1 System wide make definitions: acsMakefile

The acsMakefile sets common definitions that are valid for the whole ALMA software. It also finds out on which platform and which operating system the software is being built on and sets variables accordingly.

In addition, acsMakefile provides a set of activities normally required by every module (generation of automatic dependencies, clean, man pages, install). It knows how to deal with source code in any of the ALMA standard languages (see 7.3). The scope of such standard actions is controlled by the calling Makefile via a set of variables (see 7.2).

The variables set by acsMakefile depend on the value of some shell-environment variables (see the ENVIRONMENT section in the man pages, as explained below) and on the operating system (type and version). Make defaults can apply as well.

By default acsMakefile works for UNIX files. If the variable MAKE_VXWORKS is defined, acsMakefile works to produce VxWorks applications. (You will find this variable already defined in the Makefile_for_LCU obtained from getTemplate). This will change when the transition to Real time Linux will be completed.

To know in detail all the definitions it provides, please refer to the man pages:

```
$ man acsMakefile
```

Extensions of the acsMakefile to include new programming languages like java have been performed. To know how the java implementation works refer to:

```
$ man javaMakefile
```

Analogously, the man pages for the python extension can be accessed with:

```
$ man pythonMakefile
```


7.2 Software module Makefile

To know the structure and use of the Makefile, please refer to the man pages:

```
$ man Makefile
```

7.3 Subsystem Makefile

This is a third type of Makefile needed at subsystem level (one per subsystem) where basically the list of modules belonging to the subsystem is maintained. Refer to section 8.2 to know more. A template for the subsystem Makefile (also called, sometimes

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superseded, Obsolete)</i> Page: 14 of 43
---	---	--

package Makefile, depending on the subsystem structure) can be accessed through the utility `getTemplate -> code -> Makefile_for_PACKAGE`.

7.4 Coding standards

Again the starting point is the utility `getTemplate` with the option “code”.

After this choice, a set of templates are available per each type of supported programming language. Among others:

c-main
c-procedure
h-file

java-class-interface

c++-file
c++-small-main
c++-class-file
c++-h-file

pythonModule
pythonScript

script
tclScript

Makefile_for_WS

Makefile_for_PACKAGE

Makefile_for_LCU


RELEASE_NOTES

Every time you have to write a new piece of code, start from one of the templates. The coding standards have been defined for C, C++, IDL, Java and Python. The starting point is the web page:

<http://www.eso.org/projects/alma/develop/alma-se/reference/CodingStandards.html> .

And for official documents, the sitescape page:

<http://almaedm.tuc.nrao.edu/forums/alma/dispatch.cgi/f.702010codin>

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superseded, Obsolete)</i> Page: 15 of 43
---	---	--

The document in [2] paragraph 8 gives a general overview about the standards to be followed.

As a general rule, the names of files which get installed in common areas (INTROOT or ACSROOT), like libraries, jarfiles and the like, should be named prefixing the module name to the file. This will make it a lot easier to find out which file belongs to which module and then subsystem and developer. Very useful in a distributed development environment like that of the ALMA project!

8 CVS and how the subsystem software should be organized

As stated in [2], “Software produced by the ALMA Computing group must be stored in a centralized archive and accessible by each group member. It must be possible to track and identify previous system configurations.”

The Configuration Management Tool chosen for the ALMA project is CVS. Refer to [2] chapter 10 for a general description. CVS manuals can be found at the web sites in [7] and [8].

To access the archive (where all the software modules are stored) you need to have a login and password. Just send a mail to alma-sw-semgr@nrao.edu.

Here follows some important information about the CVS usage. The same information can be found in [9], a document written for regulating the transition from the former ALMA Configuration Management repository to CVS. (If you were using the former repository and still have to move to CVS, you may be interested in reading the entire document in [9].)

Further extensions of this manual can be foreseen to give a quick CVS tutorial. But a good quick reference for CVS can be found on the web at the link reported in [11]

8.1 The Repository Structure

The repository structure reflects the subsystem division of ALMA Computing. Expected are all uppercase subdirectories for each subsystem as in:


ACS/ ... all the modules in ACS

SCHEDULING/ ...

CONTROL/ ...

PIPELINE/ ...

ARCHIVE/ ...

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superseded, Obsolete)</i> Page: 16 of 43
---	---	--

Subdirectories may be used when deemed necessary, for instance:

```
CONTROL/Antenna/Mount
                /VA
```

```
CONTROL/AMB/Can
                /Sim
                /tp816
```

Each of the leaves of this directory tree will be a module (as defined in 6.1), with its structure and standards unchanged. An exception must be made for the directory `config` placed immediately after the subsystem.

```
CONTROL/config
```

This directory is neither a module nor a module container. It is used to deliver CDB configuration data to ITS subsystem. For its structure definition see naming convention <http://almasw.hq.eso.org/almasw/bin/view/SE/NamingConventions>

A special area called ICD has been created with the following structure:

```
ICD  /ARCHIVE
     /CONTROL
     /CORR
     /EXEC
     /OBSPREP
     /PIPELINE
     /SCHEDULING
     /TELCAL
     /Makefile
```

The ICD/Makefile is the responsibility of the ITS group. Every other subsystem will take care of populating the corresponding subsystem area with all the necessary interfaces (includes, idl files etc.).

8.2 Subsystem Makefile



ALMA Project
Title
ALMA Software Development Tools and
Integration Guidelines

Doc # : COMP-70.80.00.00-002-L-GEN
Date: 2005-05-06
Status: Draft
(Draft, Pending, Approved, Released, Superceded, Obsolete)
Page: 17 of 43

In the top level subsystem directory there will be a Makefile (for example CONTROL/Makefile) which controls the build process and which hands down the control to the various module Makefiles. “Build process” means to prepare a procedure (based on “make”) which knows all the modules belonging to the subsystem directory and goes recursively in every module to run “make all man install” for compiling and installing the software.

Notice that a subsystem may also consist of one single module, in which case it may directly have a src directory, but it must have a Makefile at the top of the tree.

To ease the work of the developers in each subsystem, a simple subsystem Makefile has been put in place (you can download it from <http://websqa.hq.eso.org/downloads/Makefile>) which is good enough to build a subsystem. The Makefile itself contains a list of all subdirectories (modules) which themselves contain Makefiles.

This subsystem Makefile implements a target called “build” which goes iteratively and in the appropriate order to each module and carries out a “make clean all install”.

Running “make build” the overall software belonging to the subsystem will be compiled and installed.

The list of modules and their order is determined by setting an internal variable (MODULES) in the subsystem Makefile. It is the responsibility of the subsystem leader to make sure that the top Makefile is updated, archived and tagged for each release (see later in this document how to prepare a release, paragraph Error! Reference source not found. and following ones.

In particular, the variable MODULES shall be kept up-to-date by the subsystem leader: new modules should be added to the list, obsolete ones removed and so on. Please notice that there can be only one default behavior concerning whether a Make command should stop when error occurs. The default will be to stop, and it will be up to the local module Makefiles not to return an error code when this is considered necessary.

There is no need for a change to the existing module Makefiles. They will continue to comply with existing acsMakefile standards.

Besides build (explained above) the subsystem Makefile will provide the following default targets:

`clean` which loops over the modules and does a clean-up (basically removing what has been created with “make all”) and similarly

`all` which loops over the modules and compiles the source files, generating binaries and necessary libraries



`install` which loops over the modules and installs in the `$INTROOT` area the binaries and libraries built with “make all”.³

`test` for running in automatic way the sequence of unit tests which have to be delivered together with the subsystem software.

The implementation of this target is based on TAT (see next chapter) and running “make test” one will perform a loop on all the modules belonging to the subsystem with the following actions:

“cd” to the directory test of every module
the command “`tat -v -nc`” will be issued.

“make test” can be run at subsystem level, to perform in one go all the tests on the subsystem, or at module level, going into the directory test of a single module. In this case, the test related to that module only (also called “unit test”) will be executed.

The following chapter explains how to write the unit test and the tool that can be used to organize the tests suite in a simple and repeatable way.

9 tools for automated testing (TAT)

For each written piece of code, a corresponding test must also be written. The test will consist of one or more programs that can be repeatable on whatever machine (and whatever supported operating system) by whatever user. But first of all, the test suite has to be run by the developer before archiving the software with the new piece(s) of code.


At ESO a tool has been developed to help organizing the test suite in a standard way, easily repeatable by an independent user, namely the integration user⁴.

This tool is called **tat**. A manual has been written explaining all the features of this tool. Please refer to [3]. Here follows a quick tutorial about `tat`.

9.1 Overview

³ Note that “install” and “all” are not expected to be used, because the actions performed by those two targets are already taken into account into the target build.

⁴ The integration user is responsible for integrating all the produced software within the ALMA project and preparing a software release (basically twice a year). His work is based on periodical integration loops during which all the new software is identified and retrieved from the archive, compiled and tested. For this it is extremely important to organize in simple and repeatable scripts all the available test programs. The final goal of this work is to validate the software release before delivering it to the different sites belonging to the ALMA project and in the future to the Observatory.

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	<p>Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superceded, Obsolete)</i> Page: 19 of 43</p>
---	--	---

The Tools for Automated Test (tat) is a framework which helps running a test suite with only one command and reports the result of the test suite in a simple and clear way: the word PASSED or FAILED is printed to the standard output for every test belonging to the suite as well as the final result, PASSED or FAILED (it's enough that one test in the suite fails to make the entire suite fail).

`tat` can be easily integrated with an existing test structure. This test structure can be a collection of scripts/programs in whatever language (supported within the project) which provide an output that has to be compared with a reference output.

The test structure has to be prepared in a software module under the directory `test` and the Makefile there has to be compliant with the project SE standards. Using this Makefile, executables and scripts will be compiled and installed in the `bin` directory of the module. They should be local. What this means is that they get installed under the `bin` directory of the module only. In the Makefile they are individuated using the suffix “`_L`”.

9.2 Limitations and future developments

`tat` handles tests based on batch programs, not interactive, and without manual interventions through GUIs, although it is possible to use GUIs whenever this is automatic (i.e., the GUI starts and exits automatically).

Starting with ACS 3.0.1, `tat` has been merged with the functionality provided by the program `eccsTestDriver`.

9.3 How TAT works

The test driver is the conductor of the automated tests. It does - almost - everything: its features provide a good overview of the test software.

The test driver:

- looks in the test directory of a module for a file named `TestList` or `TestList.lite` (the two names are equivalent) specifying initial and final scripts, environments, and test programs;
- the environment variable `SHLIB_PATH/LD_LIBRARY_PATH` is redefined to module `lib` directory first;
- the environment variable `PYTHONPATH` is redefined to module `lib/python` directory first;
- the environment variable `PATH` is redefined to module binaries first;



- executes “make clean all” to generate the test software;
- source a Tcl script containing the environment variables needed for the test (if it exists);
- executes a prologue script (if it exists);
- executes the test and compares the output with the test reference;
- executes an epilogue script (if it exists);
- executes “make clean”;

The standard output and the standard error of test scripts are first processed by :

- `grep -v -f TestList.grep <output>`
- `sed -f TestList.sed <output>`

where `TestList.grep` and `TestList.sed` are files prepared by the user with the syntax of `grep` and `sed`, used to filter the test output before comparing it with the reference.

After filtering, the output is redirected into

```
./tatlogs/run<process_id>/<testName>.out
```

If this file is the same as `./ref/<testName>.ref`, `tat` prints

```
"TEST <testName> PASSED"
```

otherwise `tat` prints

```
"TEST <testName> FAILED".
```

The reference files have to be prepared by the user, using a special `tat` command (see below 9.4).

The `TestList(.lite)` file can contain four types of directives:

SOURCE, PROLOGUE, <Test_program_name>, EPILOGUE.

The `<Test_program_name>` is the only mandatory directive; you can have one only or many test programs directives to be executed. The other directives are optional. `SOURCE`, `PROLOGUE` and `EPILOGUE` must be unique. (A fifth directive is “ENVIRONMENT” but it will not be treated in this tutorial because not used in the ALMA project)



Here is an example of a TestList.lite file (blank lines are allowed, comment lines start with "#"):

```
# directive SOURCE needs the name of a tcl script to be
sourced

SOURCE tcl_script

# directive PROLOGUE: a program or script is executed
before the # actual test

PROLOGUE myPrologueScript arg1 arg2

# TEST directive: two syntax are allowed, the first one is
#recommended, the second one is simpler and maintained for
#backward compatibility

#<testNum> <testName> <proc 1> [<proc 2>..<proc n>]

# Just a <testName>, the name of a program or script
existing in #the test directory

1 testOne "testInterface1 15" testInterface2

ccsTat

# directive EPILOGUE: a program or script is executed at
the end # of the actual test

EPILOGUE myEpilogueScript arg1 arg2
```

The **SOURCE** directive will be followed by the name of a script written in Tcl language containing environment variables to be sourced at the very beginning. An example: a test needs to work with an INTROOT, doing every sort of dirty operation there. The developer normally has an INTROOT defined in his or her environment but doesn't want to change it with the test. With the Tcl script to be sourced, a different INTROOT can be defined for the test time frame that can then be removed at the end of the test (the 'rm -rf \$INTROOT' command can be put in the epilogue script, for instance).

Note of caution about SOURCE directive and MODROOT.

The **MODROOT** variable is set immediately after the **PROLOGUE** execution and it remains set at least during the entire execution of **TESTs** and **EPILOGUE**.



ALMA Project
Title
ALMA Software Development Tools and
Integration Guidelines

Doc # : COMP-70.80.00.00-002-L-GEN
Date: 2005-05-06
Status: Draft
(Draft, Pending, Approved, Released, Superceded, Obsolete)
Page: 22 of 43

If you need to use **MODROOT** in the **PROLOGUE** simply add this line to your TestList.lite

```
SOURCE alpha.tcl
```

where alpha.tcl is a Tcl script with this content

```
global env  
  
set env(MODROOT) $env(PWD)
```

The **PROLOGUE** directive is followed by the name of a script or executable to be launched **before** the environment creation and **after** ‘make all’. The program can have one or more arguments, passed as single words separated by blanks. It can be useful, for example, to start only once processes which will be used by the different test scripts. (With the EPILOGUE script, see below, you can stop those processes).

The **TEST** directive is the only mandatory one.

A) The first type:

```
<testNum> <testName> <proc 1> [<proc 2>..<proc n>]
```

<testNum> is the number assigned to the test. Usually every record in the TestList(.lite) has a different number so that the different tests can be executed individually, just calling “tat <testNum>”.

<testName> is a symbolic name used to identify the test

<proc 1>... is a list of programs to be executed in order to perform the test. It is required to have at least one program. All programs in the list are executed in background, except the last that is executed in foreground. To pass parameters to the programs it is enough to write the complete command line for the program <proc n> in quotes.

For instance:

```
1 MyTest "test -a dfs" test_slave
```

Instead of a program to be executed, it is possible to give the special directive

```
@SLEEP n
```

where n is a number of seconds to wait before going on with the parsing of the line.



Typically this is useful when the first process in the list performs some initialization (for example writing initial values in the database to make deterministic the execution of the other components of the test. Example:

```
1 MyTest initDb "@SLEEP 5" "test -a dfs" test_slave
2
```

This means that when `initDb` is executed, a sleep of 5 second is also executed together with `initDb`.

To summarize:

- All processes you enter in a TestList record are executed in BACKGROUND, but the LAST ONE, which is executed in FOREGROUND
- When the last process (the one in foreground) exits, all other processes in the record are killed (if still alive) to cleanup the test.
- It is possible to put "@SLEEP xx" directives between any two processes to give the one on the left some time to initialise cleanly (in background) before the one on the right is started.
- @SLEEP should NEVER be the last directive in a TestList record.
- The last action in a record shall always be a command/script that drives the test.
- If you need to wait some seconds at the end of a test you can use the option `-w num` (which is valid for every test in the TestList, though).

B) The second type is a line containing the name of an executable command to be called. This command cannot have parameters and identifies the test, in the example:

```
ccsTat
```

The number of TEST directives is unlimited.

A test may call any program, script, or utility provided that this utility is not interactive and is portable among supported platforms: these prerequisites ensure that the test output is repeatable and may be used to make a test reference. The test driver calls the single test and records the output in

```
<module_name>/test/tatlogs/run<pid>/<testName>.out .
```

The output may also be filtered to remove non repeatable output (like hostnames, usernames etc.) by the test driver. To do this you need to create two files (only one of the two or both) based resp. on the `grep` and `sed` UNIX commands. This is explained in 9.6.2. Note that when using the syntax of the first type, the output will be automatically filtered against the timestamps, so you do not need to bother about looking for dates and times and preparing the TestList.grep accordingly. If, on the other hand, you already did it, it



ALMA Project
Title
ALMA Software Development Tools and
Integration Guidelines

Doc # : COMP-70.80.00.00-002-L-GEN
Date: 2005-05-06
Status: Draft
(Draft, Pending, Approved, Released, Superseded, Obsolete)
Page: 24 of 43

should not make any difference. Another characteristics of the first type of tests, is that, using that syntax `tat` will not take into account the order of the lines in the output files. (You can force `tat` to respect the order using the option “`-order`”, see `tat` man pages coming with ACS 3.0.1 or later)

A special case is when the test driver is instructed to build the reference file: in this case, the output of each single test is recorded in `<module_name>/test/ref/<testName>.ref` and the test driver will automatically perform the comparison between output and reference and print “PASS” or “FAIL” accordingly: the test passes when output and reference are identical, if not it fails.

The **EPILOGUE** directive is followed by the name of a script or executable to be launched **after** the environment deletion and **before** ‘make clean’. The program can have one or more arguments, passed as single words separated by blanks.

The `TestList.lite` file is only referenced by `tat` when `tat` is executed *without* arguments, i.e. if you specify a test by name (of a `<testNum>`) on the `tat` command line, the contents of the `TestList.lite` file will be ignored, a part from the test directive itself.

9.4 Cookbook

We will see an example of a `tat` test.

Let’s suppose we have a software module called `tattest` with the standard directory structure. Let’s have a script which prints “hello” . This script is the test for our software module. It will be put under the `tattest/test` directory of the module.

The name of the script is `testscript.sh` and looks like:

```
#!/usr/bin/ksh

echo "hello"

exit 0
```

We create a `TestList.lite` under `tattest/test` which will contain the simple line

```
testscript.sh
```

First, we try our script to see whether it works or not. When we are satisfied with the result, confident that the test has a good coverage of our code, and certain that the test can be repeatable on any system by any user, we can proceed and generate the reference file.

To generate the test reference, type:



```
$ tat -g testscript.sh
```

Reference file testscript.sh.ref generated

Reference file generated.

Tat has generated the reference file testscript.sh.ref in the **ref** subdirectory of tatest/test:

```
$ cat ref/testscript.sh.ref
```

```
hello
```

From now on, the test can be repeated just by typing:

```
$ tat testscript.sh
```

```
TEST testscript.sh PASSED.
```

```
PASSED.
```

At this point, when the test (or test suite) is finished, TAT will automatically compare the result of the actual test with the reference (the output file(s) under the ref directory) and will print the result on the console. If the actual output is identical to the reference, the test is successful (PASSED). If not, it failed (FAILED). (If you have a test suite, made up of different test scripts, it is enough that one test fails to give the final result FAILED.)

One can add as many scripts/programs as wanted to TestList.lite. If we have another script called testagain.sh which prints "goodbye", we add it to TestList.lite as the second line.

Then we generate the reference file:

```
$ tat -g testagain.sh
```

Reference file testagain.sh.ref generated

Reference file generated.

Next, we run the tat test:

```
$ tat testagain.sh
```

```
TEST testagain.sh PASSED.
```



ALMA Project
Title
ALMA Software Development Tools and
Integration Guidelines

Doc # : COMP-70.80.00.00-002-L-GEN
Date: 2005-05-06
Status: Draft
(Draft, Pending, Approved, Released, Superseded, Obsolete)
Page: 26 of 43

PASSED.

Then we can also run the suite of tests (two in our example) with a unique command:

```
$ tat
```

```
TEST testscript.sh PASSED.
```

```
TEST testagain.sh PASSED.
```

```
PASSED.
```

(Note that this is the case in which the TestList.lite file contents are referenced.)

When the option “-v” is passed to tat (tat -v), the test(s) is run in verbose mode and more information is printed to the console.

When the software module contains the two branches “ws” and “lcu”, test code can be written under both directories ws/test and lcu/test. The tat tool will compile the code under both branches and then run the test according to the TestList.lite that should be placed under ws/test only.

9.5 Commands to remember

tat

executes make, follows the actions found in the TestList.lite file (sourcing file, executing prologue script, executing the test suite, comparing the result with the reference, executing the epilogue script, running “make clean”)

```
tat <name_of_a_single_test_in_the_test_suite>
```


executes the test specified only, ignoring the contents of the TestList.lite file.

```
tat -g
```

executes the test preparing reference files under <module_name>/test/ref

All of the above commands can be run with the option -v (**verbose**) which adds more information to the standard output.

```
tat makeEnv
```

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superceded, Obsolete)</i> Page: 27 of 43
---	---	--

executes the preliminary phase of the test:

1. run 'make (clean) all'
2. source the Tcl script corresponding to the SOURCE directive (if any)
3. run the PROLOGUE script (if any)

After running 'tat makeEnv' to actually run the test one should use 'tat' and the preliminary phase will not be repeated. Only the test programs will be executed.

tat cleanEnv

This command can be used only if 'tat makeEnv' has been previously used. It executes the final phase of the test, after the test suite has been performed:

1. run the EPILOGUE script (if any)
2. run 'make clean'

9.6 Some additional features

9.6.1 Logfiles

When tat starts, it creates a directory under test, called tatlogs/run\$\$, where \$\$ is the PID of the process. In this directory whenever the test fails, the output file of the test run within \$\$ and the differences found between the output file and the reference are stored.

On every run, a directory run\$\$ is created. If a test is successful the corresponding directory run\$\$ is removed. These files can be inspected in case of test failure.

9.6.2 How to filter the test output


Sometimes the output of a test contains records that are not exactly reproducible elsewhere (for example usernames, hostnames, etc.).

In this case, it is possible to filter out those types of records using the syntax of `grep` and `sed`. `tat` gives the possibility of using two files (only one of the two or both), called:

`TestList.grep` and `TestList.sed`

When these files are present under the test directory, the standard output and the standard error of the test scripts are first processed by:

- `grep -v -f TestList.grep <output>`
- `sed -f TestList.sed <output>`

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	<p>Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superceded, Obsolete)</i> Page: 28 of 43</p>
---	--	---

Then they are redirected into ./tatlogs/run<process_id>/<testid>.out (or ref/<test_script_name>.ref).

Each line of TestList.grep will correspond to the word(s) that have to be grepped out before generating the reference or output files.

TestList.sed will contain records written in sed (a regular expression UNIX program) syntax.

9.6.3 Integration of TAT with JUnit, CppUnit and PyUnit

ITS is preparing examples for unit tests in the three supported programming languages, as well as examples of integration of tat with xyUnit tools. Work is still in progress. You can follow it at the page: <http://almasw.hq.eso.org/almasw/bin/view/ITS/SubsystemTests>.

10 Monthly Integration and ALMA Software Releases

In this chapter we will describe how the work of the ITS team has been organized to produce the ALMA software release and what the team expects from the developers to perform the integration work.

10.1 Prerequisites

As stated in [10], some external requirements to make possible the integration work have been individuated. Briefly I repeat here the basic ones:

- Every subsystem must arrive at center (at the ITS team) already integrated with ACS⁵.
- Unit tests should be prepared at module level within the framework/tool indicated by the Software Engineering subsystem.

10.2 Integration Periodicity

The Integration team will periodically perform integrations of the produced software and regression tests repeating the unit tests on the integrated system. A periodicity of performing the integration and the test tasks has been established: **once a month** all the subsystem software will be integrated at center level on an independent platform by an independent user.

The monthly integration of the subsystem software will be done the first day of each month.

⁵ The latest release of ACS at the latest patch level is the applicable one.



ALMA Project
Title
ALMA Software Development Tools and
Integration Guidelines

Doc # : COMP-70.80.00.00-002-L-GEN
Date: 2005-05-06
Status: Draft
(Draft, Pending, Approved, Released, Superseded, Obsolete)
Page: 29 of 43

At the Santa Fe meeting, (17-20 November 2003) it has been decided that the interfaces will be frozen the 15th of every month, in order for the developers to develop, in the following 15 days, software based on stable interfaces before delivering the software itself to the ITS group. The ITS group will then try to integrate the interfaces only (the ICD directory structure created in CVS) the 16th of every month.

Another type of integration with periodicity 1 month (beginning of every month) has also been assigned to ITS: this is the integration of the ACS release + ARCHIVE under development. This is because ACS+ARCHIVE are the basic layer used by every developer in whatever subsystem for his/her daily work. So it is crucial that the ACS+ARCHIVE software be stable. The ACS+ARCHIVE software is for this reason released at the very beginning of every ALMA SW release cycle, so that the other subsystems have enough time to install the new software and base the development for the ALMA SW release on that stable ACS+ARCHIVE software. That software will not change during the preparation of the ALMA SW release (unless a patch is required). In the meanwhile, ACS+ARCHIVE developers can begin to prepare the following ACS+ARCHIVE release and ITS has the task of retrieving that code at the beginning of every month, integrating it and testing it. A special CVS tag (different from the one used by all other subsystems' developers) for those two subsystems has been agreed, in order for ITS to retrieve the proper software.

10.3 Integration and Testing Strategy

To properly perform the integration task in a big software project like ALMA, an integration strategy should be established (which subsystems should be integrated first? what order should be used to integrate subsystems?).

The integration order has been established as follows: ACS ICD ARCHIVE CORR OBSPREP TELCAL CONTROL PIPELINE SCHEDULING EXEC. Note that the order does not have to be considered frozen: it can change in the future according to the needs of the ALMA software and the requirements from the different subsystems.

ACS and ARCHIVE (as explained above in this chapter) will not change during the integration cycles of the ALMA software release. They will instead be integrated separately in a separate release cycle.

The ICD will be integrated fifteen days before the rest of the subsystems to prove that the interfaces can be compiled together without conflicts.

An incremental approach to perform the integration tests should be used, starting from a small group of subsystems and increasing gradually the number and complexity up to the overall system.

For example, for R1.1 we used the strategy of testing first the subsystems in isolation (besides the unit tests, also some functional tests on the single subsystem have been performed, simulating when possible the interfaces with other subsystems), then a group of subsystems involving



ALMA Project
Title
ALMA Software Development Tools and
Integration Guidelines

Doc # : COMP-70.80.00.00-002-L-GEN
Date: 2005-05-06
Status: Draft
(Draft, Pending, Approved, Released, Superseded, Obsolete)
Page: 30 of 43

SCHEDULING, CONTROL, PIPELINE and TELCAL has been chosen (with CORRELATOR in simulation).

A first system test has been also performed (not yet completed for some problems with CORRELATOR).

This work should be continued in this way, adding more and more test cases, with the next ALMA SW releases.

10.4 Integration procedure

To perform the integration task a standard method to build the software in an automatic way has been implemented. As explained in chapter 8, the software configuration management tool for the ALMA project is CVS. The build procedures take into account the CVS configuration and use the Makefile and the “build” target (as explained in chapter 8) to perform the integration automatically.

A script loops on all subsystems, retrieving the tagged software from the CVS repository, (see below the paragraphs 10.8 and 10.9 to know more about tags) executing the subsystem Makefile and compiling all software modules belonging to each subsystem. The results are saved in log files that can be subsequently inspected.

Whenever a bug fixing in a module is needed, the responsible person for that module is supposed to provide the ITS team with a new tagged version of the software. ITS will then repeat the retrieval from the CVS server and the build of the software.

10.5 Testing procedures


The ITS team is supposed to execute different types of tests, starting from unit tests, then integration and functional tests, finally the overall system test. (Integration and system tests can also include performance and stress tests).

For running unit tests, the tool at the moment used is TAT (Tool for Automated Testing). It can be called directly or through the Makefile target “make test”.

At the end of the build procedure, the unit tests are executed using an automatic script which executes “make test” in all software modules, subsystem per subsystem. As for the compilation, the results are saved in log files.

While each developer has to provide unit tests for the software he/she wrote, the ITS team prepares the integration and system tests, using inputs from subsystem developers or SSR people.. Following the strategy in 10.3, the tests are prepared starting from the requirements and deliverables for the release under preparation. Test cases are written as a step by step sequence of actions to perform, in order for everybody to be able to repeat the test. The test cases are put in a procedure with the aim to make it as automatic as possible.

All procedures are under configuration control in the ITS area under CVS.

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	<p>Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superseded, Obsolete)</i> Page: 31 of 43</p>
---	--	---

10.6 Tool(s) for Dynamic Analysis

The ITS team also performs dynamic analysis on the delivered software, using coverage analyzers and memory leak detectors. Some experiments with Purify and Coverage have already been put in place. The results are published on the web under the ITS web site after every monthly integration or release.

10.7 Releases

The first (pre)-release (R0) for the ALMA software has been prepared for 2003 May 1. After R0, an ALMA software release has to be produced every six months up to T1 (30 May 2007) (see [4] for a complete overview of the milestones of the project).

The releases up to T1 are called **subsystem releases** and are identified with the names:

RX.0 and RX.1, where

X = 1, ..., 4 and

0 identifies a major release (foreseen for the first of October every year) and

1 identifies a minor bug fixing release (6 months after the major release).

These releases will be based on the latest available (and stable) ACS release.

The subsystem releases are completed when all code is checked in, builds cleanly and passes all "unit" tests for functionality planned for that release.


The ITS group will deliver the **integration release** (containing all the integrated software of all ALMA subsystems) within two months after delivery of subsystems' software. This means the 30th of November every year for the major releases and, for the bug fixing releases, the 31st of May.

The ALMA integration release is completed when all ITS tests for functionality planned for that release pass.

10.8 CVS tags for the monthly integration

To produce the integration release, the ITS group will regularly perform **monthly integrations** retrieving the software from the CVS repository.

As a consequence, once a month the developers must archive the versions of their software modules which pass unit tests.

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	<p>Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superceded, Obsolete)</i> Page: 32 of 43</p>
---	--	---

The software in CVS for the monthly integration will be tagged with a conventional tag,

MONTHLY-YYYY-MM,

where MM is two digits for the month (example: January -> 01, February -> 02 etc.) and YYYY the year. In this way, all the software with that tag will be retrieved by the ITS team and integrated during the integration at month MM. Note that when working with CVS the TAG format has to be decided within the group and in principle can be whatever. Only the monthly tag has a standard format that has to be respected by all the subsystems.

The deadline to archive the software in CVS using the tag MONTHLY-YYYY-MM is the last day of every month.

With the same tag, the 15th of every month the developers should archive and tag in CVS the ICD/<SUBSYSTEM_NAME> they are responsible for.

The monthly tag MONTHLY-YYYY-MM is the one expected by the ITS group to perform the monthly integration. Every subsystem leader will coordinate within his group the work of the developers in order to supply to the ITS subsystem the tagged ICD software by the 15th of the month and the rest of the software by the end (last working day) of each month.

For the monthly ACS+ARCHIVE integration, ACS and ARCHIVE will have to be tagged with the agreed CVS tag:

ACS-n_m_p-pre-YYYY-MM

Where:

- n is the major number of the ACS release under preparation
- m is the minor number
- p is the patch level
- YYYY is the 4-digit year
- MM is the two-digit month

10.9 CVS tags for milestone releases Rn

For the sake of simplicity, we decided that also the software to be delivered for a major or bug fixing ALMA software release will be tagged with the same monthly tag:

MONTHLY-YYYY-MM

After that, at release milestone, the code will be frozen: no modification will be allowed any longer on the released software, unless explicitly required by the ITS group for bug-fixing



ALMA Project
Title
ALMA Software Development Tools and
Integration Guidelines

Doc # : COMP-70.80.00.00-002-L-GEN
Date: 2005-05-06
Status: Draft
(Draft, Pending, Approved, Released, Superceded, Obsolete)
Page: 33 of 43

purpose. This will give the possibility to the ITS team to perform the integration tests and deliver the ALMA software release.

In case a bug fixing is necessary, the correct procedure to follow is:

First, retrieve the tagged version:

```
$ cvs co -rMONTHLY- YYYY-MM subsystem_name
```

Then, open a branch on it:

```
$ cvs tag -b MONTHLY- YYYY-MM -B subsystem_name
```

MONTHLY- YYYY-MM -B being a conventional name for naming the branch.

(Strictly speaking, it doesn't really matter when you issue these commands. Once the original Rn tag has been assigned, you can always carry out the above steps.

We think though, that since the likelihood that bugs exist and will need to be fixed is high, it is good practice to execute the above commands immediately after the original tag MONTHLY- YYYY-MM has been put).

Then to actually work at the bug-fixing you have to make your working copy the branch, so (after removing previous working copies) you should issue the command:

```
$ cvs co -rMONTHLY- YYYY-MM-B subsystem_name
```

There you can work, make modifications, check in the software.

When you are ready, after every fix, the subsystem software shall be tagged again with MONTHLY-YYYY-MM-n

where n is a progressive number starting with 1 till infinite (hopefully not too much bigger than 1...).

So, for the first bug fixing you will issue the command:

```
$ cvs tag MONTHLY- YYYY-MM-1 subsystem_name
```


and inform ITS.

Analogously for successive modifications (bug fixing) to the code, you will start from the branch with:

```
$ cvs co -rMONTHLY- YYYY-MM-B subsystem_name
```

...modifications, check in.... and tag when ready:

```
$ cvs tag MONTHLY- YYYY-MM-2 subsystem_name
```

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	<p>Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superceded, Obsolete)</i> Page: 34 of 43</p>
---	--	---

and so on.

Bear in mind that after all these commands have been issued, in order for you to continue the development on the main trunk you will have to

```
$ cvs co -rHEAD subsystem_name
```

first.

This way of working will let you develop without problems on both the branch and the main trunk.

If then you need to merge the modifications done in the branch with the main trunk you can use the command

```
cvs update -j
```

The integration process may take up to two months. During this process, the development may go on but the subsystem leaders are not supposed to tag the software with the monthly tag. (Of course, if necessary for the developers, they could tag with other types of tags).

When all the subsystems' software has been integrated and has passed the unit tests as well as the integration and system tests, an overall tag will be put on that software, using the format: ALMA-RX_0 or ALMA-RX_1.

10.10 Integration hardware


The integration and test activity is performed in both sites (Garching and Socorro) on a set of integration machines different from the development machines and by an independent user. At the same time, a Standard Test Environment (STE) is being prepared. The STE will dictate the standard configuration for Linux and RTOS machines. As soon as the STE will be ready, we encourage the developers to test their release against this set of machines. Information about the STE can be found at the page:

<http://almasw.hq.eso.org/almasw/bin/view/ITS/StandardTestEnvironment>

10.11 Reports

After the monthly integration, a report will be sent to management via e-mail. The web site at <http://www.eso.org/projects/alma/develop/software/alma-it/> has also been prepared. There, all the monthly reports are collected.

The reports will contain information about:

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	<p>Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superceded, Obsolete)</i> Page: 35 of 43</p>
---	---	---

- portability issues and compilation problems found
- test coverage and other metrics
- test results

10.12 Distribution of the integrated software

During the preparation of every ALMA SW release, it is important that every developer have the possibility to compile and test his/her software against the rest of the software. In this way, he/she will quickly find out about naming conflicts or other types of conflicts that otherwise will be discovered only at the monthly integration.

For this reason, after every monthly integration, the compiled and test ALMA SW will be made available to all the ALMA SW developers. A tar-gzipped file of an ACSROOT (containing not only the ACS software but the compiled integrated software from all subsystems) will be put on the ITS website, in a password-protected page (see <http://www.eso.org/projects/alma/develop/software/alma-it/> under the title “Download”).

All the developers are invited to download and install the tar-gzipped file. The best way of doing is: to save the “pure” ACSROOT under the current ACS release and untar the file at the place of the normal ACSROOT. If the current ACS software release is ACS 2.1, one should have the following directory structure under /alma/ACS-2.1:

ACSROOT.pure

ACSROOT #this one contains the entire ALMA SW

Java

Gnu

JacORB


...

At the following monthly integration, a new tar file will be available for download. It is recommended to save the previous one in ACSROOT.old and install the new one.

In this way, it will be easy to switch among ACSROOT.pure, ACSROOT and ACSROOT.old whenever the exigency of debugging problems arises.

10.13 Working Areas organization and new directory structures

At the time of writing, the only area available for installing the software is the ACSROOT, besides the INTROOT which should be owned by the developers only.

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	<p>Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superceded, Obsolete)</i> Page: 36 of 43</p>
---	---	---

This causes practical problems at development centers, where the subsystem software is developed. Every subsystem is supposed to have the ACS software installed in the ACSROOT. But then, every subsystem will also produce a software package, for example TICS. Every developer, at a certain point of the software cycle, has the exigency to access the overall software package integrated somewhere, possibly in an area different from ACSROOT and his own INTROOT.

Because of that, at development centers the need for separated working areas like TICSROOT, OTPROOT, ..., arose. For satisfying this need, a new variable called INTLIST has been introduced in the set of environment variables as well as in the acsMakefile and delivered with ACS 4.0. Information can be found at the page: <http://almasw.hq.eso.org/almasw/bin/view/ITS/INTLIST>. Briefly, INTLIST should be defined at user level as follows:

```
INTLIST=      $TICSROOT:$OTPTROOT
```

where the first XXXROOT has precedence (in the PATH and LD_LIBRARY_PATH) on the following ones.

For the integration of the overall ALMA software, after R2.1, ITS will modify the integration procedures in order to use INTLIST and create the following separated areas:

- ACSROOT (containing ACS+ARCHIVE)
- ICDROOT (containing the ICDs)
- ALMAROOT (containing the rest of the software).

Three corresponding tar files will be made available at the ITS Download site.

11 Software Problem Report (SPR) system

11.1 Overview


The SPR system is built using the commercial tool Action Remedy (c) and has a Web Browser interface⁶.

The SPR System should be used by internal or external users of the ALMA software in case an error in some code or in the documentation is found or a change to improve the behavior of the existing software seems needed.

Access is granted to registered users only via a login procedure.

To get a login send a mail to alma-sw-semgr@eso.org with the following information:

⁶For the Netscape users: please, note that the actual version of Action Remedy (4) does not support yet browser versions later than Netscape 4

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	<p>Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superseded, Obsolete)</i> Page: 37 of 43</p>
---	--	---

- Login name
- Full Name
- Institute address and phone number
- Full email address

Refer to alma-sw-semgr@eso.org for any other problem/question concerning the system.

11.2 Login into the system

Run Netscape (or another Web browser) connect to the page:

<http://support.eso.org/arweb/C/arweb.jsp?Form=login.jsp&>

The login page is shown. Provide your identification in terms of username and password.

If you have problems accessing the system or you have forgot your password contact alma-sw-semgr@eso.org

11.3 Submitting an SPR

To submit an SPR use the following procedure:


- login into the SPR system
- choose the link "New" for the ALMA SPR schema
- the submit page should be displayed. Fill it as suitable. (You can also use the on-line help). The fields whose labels are displayed in bold are mandatory, the others optional.
- click "Submit". The SPR will be added to the archive. The initial status is "Open".

Note: Please remember to specify the Consortium/project/site you belong to on the Location field

11.4 Querying the system

The query functionalities are available using the following procedure:

- login into the SPR system
- choose the link "Find" for the ALMA SPR schema

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	<p>Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superseded, Obsolete)</i> Page: 38 of 43</p>
---	--	---

- the Search page should be displayed

From there on is possible to perform the following main operations:

- Get a list of SPR

The selection can be performed using one or more of the available fields.

Once the searching rules are set, click on “Run Search”. As result of the selection a list of the SPR(s) which match the search criterion is displayed.

- Display a specific SPR

Specify the SPR number and press “Run Search” or if the SPR appears already on the list of displayed SPRs (obtained at point 4), simply clicking on “Display” for the desired SPR entry in the list.

- Add a Remark

To add a remark to a selected SPR click on “Modify”. The Modification page should be displayed. Add any remark on the Worklog field and click “Modify”.


11.5 Changing the login password

- login into the system
- choose "Find" for the "User" schema
- type your login in the "Login name" field
- run “Search”
- choose “Modify”
- change the password field
- click “Modify”

11.6 ALMA SPR System workflow

Please, refer to the SE page:

<http://www.eso.org/projects/alma/develop/alma-se/reference/SprOverview.html>

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	<p>Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superseded, Obsolete)</i> Page: 39 of 43</p>
---	--	---

11.7 ALMA Software Board

The ALMA Software Board (ASB) periodically meets to discuss the open SPRs. Basically decisions have to be taken whether the SPR is accepted or not, whom the SPR has to be assigned to and by when the SPR has to be solved.

The board is called under responsibility of SE. Please refer to the page: <http://www.eso.org/projects/alma/develop/alma-se/reference/ProcASB.html> to know more.

12 Documentation

The appropriate documentation has to be written together with the code.

We can identify many levels and types of documentation:

- 1) Comments inside the code
- 2) Header of each source file to generate man pages
- 3) Comment to be written when archiving a file (CVS log file)
- 4) Release Notes
- 5) Manuals

You can use your preferred editor to write comments into your code, vi and Emacs being the most common ones. For developers, Emacs offers a set of utilities and customizations to ease and improve the developer's work.

For official documents (see chapter 12.5), we use Microsoft Word. When the document is ready, a PDF copy together with the word version should be made available.

12.1 Comments inside the code

The manuals about the coding standards, (see <http://almaedm.tuc.nrao.edu/forums/alma/dispatch.cgi/f.702010codin>) contain a chapter regarding the documentation.

Please, note that the ALMA standard code documentation tool is doxygen.

Doxygen is the tool used, among other things, to automatically generate the ICD documentation. The procedure can be found at the following link:

<http://almasw.hq.eso.org/almasw/bin/view/SE/ICDDocumentation>

12.2 Header of each source file to generate man pages



Whenever you start writing a program, you shall use the utility `getTemplate`. Run the utility and choose the option “code”, you will access a bunch of templates for writing in different programming languages, among others C, C++ and Python.

Note that the initial part of the template for source files is the so called header and has to be carefully filled in. This part will be used to automatically generate the man pages for the corresponding program, when running “make man”. It consists of a series of sections.

NAME, SYNOPSIS and DESCRIPTION sections are always present and mandatory. Depending on the type of file, other sections are present, some mandatory and others optional. If any mandatory section is empty, its header is kept with one of the following: “Not Applicable” or “None”.

If any optional section is empty, its header is omitted.

Remark: the one-line description under the NAME section should contain the essential keywords to identify the functionality performed because the UNIX apropos searching feature is based on such keywords.

It is a good practice to document the changes performed on each file belonging to a module. For this, besides the header, every developer who worked at the file should add a line like the following information:

```
# who          when          what
# -----
# userXYZ      2001-07-08    created
# userIJK      2001-12-03    added support for library
```


in the initial part of the file.

12.3 CVS log file

Every time you want to check in a modification, CVS requires you to either use the `-m` option or edit a file, writing a short comment related to your modification.

12.4 Release Notes

For every major release of the ALMASW, the complete set of Release Notes for all the ALMASW applications and programs has to be produced. It is up to the subsystem leader to organize the Release Notes at package level or module level and appoint the persons who have to write them. The final file containing the release notes for the whole subsystem should be called `RELEASE_NOTES` and put under `<SUBSYSTEM_NAME>` in CVS.

	<p>ALMA Project Title ALMA Software Development Tools and Integration Guidelines</p>	<p>Doc # : COMP-70.80.00.00-002-L-GEN Date: 2005-05-06 Status: Draft <i>(Draft, Pending, Approved, Released, Superceded, Obsolete)</i> Page: 41 of 43</p>
---	--	---

A template for writing Release Notes is available from getTemplate -> code -> RELEASE_NOTES.

A procedure to collect the Release Notes at every major ALMA SW release and publish them (possibly on the internet) has still to be prepared.

12.5 Manuals

The type and number of manuals to be written during the life-time of a project have been established in a general way in [4] and, more specifically, in [2], chapter 3 and 4. For knowing more about the document revision procedure, look also at [5].

12.5.1 Templates and Document Number

The Templates for the Word documents can be found at:

<http://almaedm.tuc.nrao.edu/forums/alma/dispatch.cgi/templates/folderFrame/100015/0/def/9c9e>

while the procedure to give a document number is explained in [6]. This procedure supersedes what can be found in every other document.

12.5.2 Sitescape

Within the ALMA project an ALMA Electronic Document Management System (ALMAEDM) has been setup, based on the commercial tool Sitescape.

The starting page is <http://almaedm.tuc.nrao.edu>. You need a login to access the system. For this, send a mail to bglenden@nrao.edu.





ALMA Project
Title
ALMA Software Development Tools and
Integration Guidelines

Doc # : COMP-70.80.00.00-002-L-GEN
Date: 2005-05-06
Status: Draft
(Draft, Pending, Approved, Released, Superseded, Obsolete)
Page: 42 of 43

Appendix A. A CRASH-COURSE IN BASH SHELL

What follows is a little cookbook of common things you may wish to put in your start-up files:

1. To set a variable do this:

```
VARIABLE1=value  
VARIABLE2="value with spaces"  
export VARIABLE1 VARIABLE2
```

(Notice there are no spaces around the '=')

2. To safely test if a variable is set do this:

```
if [ "X$VARIABLE" != X ]; then  
code to do if VARIABLE is set  
else  
code to do if VARIABLE is not set  
fi
```

3. To set an alias:

```
alias ALIASNAME="alias value"
```

(e.g: alias ls="/bin/ls -a")

4. A switch statement looks like this:

```
case "$VARIABLE" in  
value1) do this  
  
and then this  
  
and this is last ;;  
value2) do this  
  
and something else ;;  
*) this is the default action ;;  
esac
```



ALMA Project
Title
ALMA Software Development Tools and
Integration Guidelines

Doc # : COMP-70.80.00.00-002-L-GEN
Date: 2005-05-06
Status: Draft
(Draft, Pending, Approved, Released, Superseded, Obsolete)
Page: 43 of 43

5. how to write a function.

You don't like to use

```
"export VARIABLE=VALUE"
```

and you want to keep using

```
"setenv VARIABLE VALUE"
```

For this, you should write a function, because aliases which require embedded parameters must be written as functions. Here's an example:

```
export_function()  
{  
export $1=$2  
}  
alias setenv=export_function
```