
Atacama
Large
Millimeter
Array

# Starting out with ACS:
## A 'new ALMA Developers' guide

ALMA-00.00.00.00-000-A-TTT

Version: A
Status: Draft
2004-04-09

Sohaila Roberts
*National Radio Astronomy Observatory*

| **Prepared By:** | | |
|---|---|---|
| **Name(s) and Signature(s)** | **Organization** | **Date** |
| Sohaila Roberts | NRAO | 2004-04-09 |
| **Approved By:** | | |
| **Name and Signature** | **Organization** | **Date** |
| | | |
| **Released By:** | | |


Atacama
Large
Millimeter
Array

| Name and Signature | Organization | Date |
|---|---|---|
| | | |

# Change Record

| Version | Date | Affected Section(s) | Change Request # | Reason/Initiation/Remarks |
|---|---|---|---|---|
| A | yyyy-mm-dd | xxxxx | ALMA-00.00.00.00-000-A-CRE | xxxxxxx |
| | | | | |
| | | | | |
| | | | | |

# Table of Contents

# 1 Description

## 1.1 Purpose

To help new ALMA programmers become familiar with using ACS. To help them know where to look and what to look for.

## 1.2 Scope

Helps developers new to ACS find what they are looking for when starting with ACS. The documents referenced herein still must be read for a complete understanding of how to use ACS.

This document is pretty elementary for a developer already familiar with ACS. Developers with ACS experience who are migrating to the Java implementation can safely jump to section 5.

This document is associated with the ACS 3.1 release. To find the documents specifically related to another release go to the ACS webpage (http://www.eso.org/~gchiozzi/AlmaAcs/) and select the release you are interested in.

# 2 Related Documents

## 2.1 References


1. ALMA Software Glossary
   http://almaedm.tuc.nrao.edu/forums/alma/dispatch.cgi/Glossary
2. CORBA-based Common Software for the ALMA project
   http://www.eso.org/~gchiozzi/AlmaAcs/OtherDocs/ACSPapersAndSlides/spie2002.pdf
3. ALMA Software Development Tools and Integration Guidelines
   http://www.eso.org/~gchiozzi/AlmaAcs/OnlineDocs/IntGuidelines.pdf
4. ALMA Common Software Installation Manual
   http://www.eso.org/~almamgr/AlmaAcs/Releases/ACS_3_1/Docs/ACS-Installation-Manual.pdf

5. ACS Java Component Programming Tutorial
   http://www.eso.org/~gchiozzi/AlmaAcs/Releases/ACS_3_1/Docs/ACS_JAVA_Component_Tutorial.pdf
6. Notification Channel Design & Tutorial
   http://www.eso.org/~gchiozzi/AlmaAcs/Releases/ACS_3_1/Docs/Notification_Channel_Module_Software_Design.pdf
7. ALMA Common Software Overview http://www.eso.org/~gchiozzi/AlmaAcs/Releases/ACS_3_1/Docs/ACS-Overview.pdf
8. ACS C++ Component/Container Framework Tutorial http://www.eso.org/~gchiozzi/AlmaAcs/Releases/ACS_3_1/Docs/BACI_Device_Server_Programming_Tutorial.pdf
9. ALMA Common Software and Python
   http://www.eso.org/~gchiozzi/AlmaAcs/Releases/ACS_3_1/Docs/ACSPython.pdf
10. ALMA Common Software and Python
    http://www.eso.org/~gchiozzi/AlmaAcs/Releases/ACS_3_1/Docs/ACSPython.pdf
11. ACS Supported BACI Types
    http://www.eso.org/~gchiozzi/AlmaAcs/Releases/ACS_3_1/Docs/ACS_Supported_BACI_Types.pdf
12. Management and Access Control Interface Specifications
    http://www.eso.org/~gchiozzi/AlmaAcs/Releases/ACS_3_1/Docs/Management_and_Access_Control_Interface_Specification.pdf
13. Configuration DataBase
    http://www.eso.org/~gchiozzi/AlmaAcs/Releases/ACS_3_1/Docs/CDB.pdf
14. ACS Basic Control Interface Specification
    http://www.eso.org/~gchiozzi/AlmaAcs/Releases/ACS_3_1/Docs/ACS_Basic_Control_Interface_Specification.pdf

# 3 Introduction

One of the best descriptions of what ACS really is and what it's good for is a document called CORBA-based Common Software for the ALMA project [2]. There is also the ALMA Common Software Overview [7] which should be read as well.

Before you begin reading any of the other documentation listed here make sure you have read and understood the ALMA Software Development Tools and Integration Guidelines [3]. The common terminology and assumptions which all ALMA documents use is defined in it.

# 4 Getting Started

## 4.1 Installing ACS

Although the ALMA Common Software Installation Manual [4] describes the formal procedure to install ACS, in practice the most efficient method is to download the binary distribution (a tar.gz file) which the ACS team provides. As the above document mentions, the default user for the installation is almamgr. Section 4.3 in that document tells you how to create an almamgr account. The default place to unpack the binary is in */alma* if you are working on your own Linux machine. However, if you are working on a public machine and don't have access to install into the root directory you can install it anywhere but there will be additional changes which you will have to make to your environment variables (see Section 4.2). Then you will need to copy the .acs directory and all its contents to your home directory (*.acs* is found in *$ACSROOT/config*). This is also described in Section 4.3 of the above document.

It should be noted that ACS 3.1 is designed for RedHat 9 but there is also a version for RedHat 7.2.

## 4.2 Setting up your environment to use ACS

Now before you begin to use ACS you need to set up your environment. In your home directory you should have copied over an *.acs* directory that contains a *.bash_profile.acs* file. You will notice that it is a bash resource file; this is because ACS is designed to run under bash. Although there are people that create resource scripts to run ACS under other shells it is suggested that you use the system in the environment that it was designed for.

In the *.bash_profile.acs* file are all the environment variables that are required by ACS. If you have installed ACS into */alma* then the only variable you will need to alter is the INTROOT (described below). However, if you have installed it into

any other directory you will have to tell the *.bash_profile.acs* file where you have installed it. Open the file and look for ALMASW_ROOT=*/alma* then change the */alma* to be the location where your ACS version is installed.

The INTROOT variable should be defined when you do any programming. INTROOT stands for integration root and you should use the 'getTemplate' script to create one, see the integration guidelines document [3] for more on this topic. Your INTROOT directory is not your working directory. The INTROOT is part of the ACS runtime search path; hence any code located there is available for use within the ACS system.

As a concrete example: Parts of the ALMA Scheduling subsystem rely on files defined in the ALMA Obsprep subsystem. So in order to successfully compile the Scheduling subsystem the Obsprep subsystem must already be installed. Rather than putting this rapidly (compared to ACS) changing code in the ACS directory, putting it in the INTROOT allows independence from the ACS distribution. The easiest way to install the files into the INTROOT is to use the *make install* target defined in the ACS makefiles.

Another great advantage of using INTROOT is that if a mistake is made you can safely delete it without having to reinstall ACS!

By default the ACS Makefile installs software which interacts with ACS in INTROOT (or ACSROOT if INTROOT is not set in your *.bash_profile.acs).* Defining INTROOT allows the developer to separate their modifications from the ACS distribution. This variable is initially commented out in the *.bash_profile.acs* file. You must uncomment it and set it to the correct location manually. See Section 6 in ALMA Software Development Tools and Integration Guidelines [3] for more information on INTROOT.

### 4.3 Sourcing your new environment variables

Now that you have a *.bash_profile.acs* file that has been edited to your specific ACS installation you need to make sure that every time you log in it gets sourced.

Unless you want to manually do it yourself in each terminal window put the following lines into your bash resource file (usually it is *.bashrc* or *.bash_profile* in your home directory)

*. $HOME /.acs/.bash_profile.acs*

This will automatically source the ACS resource file when you log in.

You can always then check to make sure it did source the file by testing some of the variables using 'echo $ACSROOT' or 'echo $INTROOT'.

## 4.4 The Makefile system

By now you've heard mention of Makefiles and they must be addressed because of their complexity. ACS has a built in Makefile system that hides most of this complexity from the subsystem developers. The reason this system is in place is because of the enormous path searching each developer would have to re-implement if they created their own Makefiles.

Makefiles are discussed more formally in ALMA Software Development Tools and Integration Guidelines [3], Section 7.

The standard Makefile template which you get from getTemplate contains a lot of tags. Only a few of these are required for Java. These include

- JARFILES=
- jjj_DIRS=
- jjj_EXTRAS=
- USER_JFLAGS =

So you can probably guess that the jjj is something that needs to be edited. An example of these tags in use would be

- JARFILES = scheduling
- scheduling_DIRS = alma
- scheduling_EXTRAS = alma/scheduling/image/*.jpg

So using the Makefile that contains these tags, the jar file that gets created when the system is built is called scheduling.jar, as defined by the JARFILES tag. The

directory for its source code is contained in ./alma as the scheduling_DIRS tag specifies(*note: this example is for the Makefile that will be located in the src directory of your subsystem. If you are unfamiliar with this concept refer to Section 6 in the integration guidelines document [3])* and any extras, such as JPEGs, that you want included in your jarfile would be found in ./alma/scheduling/image/ as specified by scheduling_EXTRAS tag.

The USER_JFLAGS tag is quite useful because this is where a Java developer would put the any options they wish to send to the Java compiler. One great example would be

- USER_JFLAGS = -deprecation

If you need to more details about the Java Makefile type 'man javaMakefile' in a terminal window on Linux.

Another tag of interest is DEBUG. If this tag is set to equal 'on' then when the jarfile is created it not only adds the class file but it adds the java source files too. This is extremely useful if your code does any code generation (For example: XML to Java or IDL to Java)!

For C++, some of the makefile targets that are important to you include

- INCLUDES =
- LIBRARIES =
- lllll_OBJECTS =
- lllll_LIBS
- USER_LIB =
- USER_INC =

The INCLUDE target is where you would specify any public header files. The files specified by this target just get copied to your $INTROOT when you do a make install.
The LIBRARIES target is where you specify the library that your implementation files will be compiled. For each file specified in the LIBRARIES target you need a corresponding lllll_OBJECTS target.

The llllll_OBJECTS target is where you specify all the objects that you want to generate from your implementation files. The Makefile assumes that you have a .cpp or a .C file that corresponds to each file you specify in this target. For example, if you have 2 files specified by the LIBRARIES target you will need the following for the llllll_OBJECTS target. Note the "llllll" is replaced by the library.

- LIBRARIES = foo1 foo2
- foo1_OBJECTS = foo1a foo1b foo1b
- foo2_OBJECTS = foo2a foo2b

You may also have a llllll_LIBS for each library.

- foo1_LIBS = foo1Stubs

More is explained about compiling C++ in the ACS C++ Component/Container Framework Tutorial [8], in Section 7.

In any language, though most commonly used in C++ is the target

- CDB_SCHEMA=

This target is necessary when you have defined a schema for component properties. You schema won't be installed into your $INTROOT unless you tell the Makefile it exists. See Section 7.3 for more information.

Occasionally you will run into odd behaviors that are due to remnants of previous complications. These can usually be resolved by a simple 'make clean all'.

## 4.5 Configuration Database (CDB)

After reading the documentation referenced above you've probably seen the term CDB a fair bit. The CDB is essential to making your component show up and be accessible in the ACS world. Since the CDB is so important it does get its own environment variable, ACS_CDB. Like INTROOT you can put your ACS_CDB anywhere. However, there is a defaultCDB that comes with ACS. You can edit

that one with your additions or you can create your own using the default one as a guideline.
The files you are most concerned with are

- $ACS_CDB/CDB/MACI/Components/Components.xml
- $ACS_CDB/CDB/MACI/Managers/Manager/Manager.xml

Section 4 in the ALMA Common Software Overview[7] explains more about the CDB. Also as you go through the tutorials (not just the Java ones) you will find example entries.
Two documents that describe concepts in the CDB are Configuration DataBase[13] and Management and Access Control Interface Specification[12].

If you are a programmer working with properties/characteristics, see Section 7 below, you will need to understand the CDB in greater detail. Since properties/characteristics require schemas and instance-xml files you will need to create files in $ACS_CDB/CDB/schema (for the property's schema) and $ACS_CDB/CDB/alma (for the property's instance-xml). You can see examples of this in the default CDB that is distributed with ACS. One thing to note from the C++ tutorial [8] is that the schema's location, which is illustrated in Section 3, points to ../config/CDB/schemas/. Which basically implies you have to put the schema in to the config directory of either your $INTROOT or working directory but it can also be installed into the $ACS_CDB/CDB/schemas directory as mentioned above. The reason you would want to put it into your subsystem's config directory and specify the Makefile target CDB_SCHEMA is because other people who wish to use your subsystem will need that schema to run your software. By putting your schema only into your $ACS_CDB no one else will have access to it and will most likely run into problems.

### 4.6 A little side note on container names

Now before we start up ACS to see if it's working correctly a quick mention should be made about the container names. The default names that ACS 3.1 uses for each language are: *frodoContainer* for Java, *bilboContainer* for C++ and *aragornContainer* for Python. These specific names were probably chosen because

the previous release of ACS, ACS 3.0, was released around the same time that Lord of the Rings 3 was released.
The names for these containers are arbitrary. As long as the container you start corresponds to the container specified by your component's entry in the $ACS_CDB/CDB/MACI/Components/Components.xml file your component will start properly.

### 4.7 Did you install ACS correctly? Let's check!

So now we want to start up ACS and see if it correctly connects to an example that comes with the default installation. There are two ways you can start and run ACS so both will be described here.

First we'll see it demonstrated on the command line of Linux terminal windows. Open at least 3 terminals, in the first one type 'acsStart'. This command does two things, it first starts the ORB services and then it starts the manager. If you aren't familiar with this terminology go back and read the reference documentation because it only gets more complex from this point on! (You can also run these two services separately with the commands acsStartORBSRVC and acsStartManager.)

You will notice that as soon as the script started a line was printed saying what the ACS_INSTANCE number is. When using multiple terminals to run ACS make sure that the ACS_INSTANCE variable is set correctly in each terminal. The default value is 0. The ACS_INSTANCE variable is primarily documented in ALMA Common Software Overview[7] in Section 2.2 but is also mentioned through out that whole document.

When the above script has finished (you will see the statement 'Manager is up and running') you will want to start a container for the Java component. If you look in the Components.xml file in your CDB you will see which container the component uses. As mentioned in Section 4.5, in the default CDB all Java components use the *frodoContainer,* all C++ components use *bilboContainer,* and all python components use *aragornContainer.* Since we just want to try a Java

component we will only start the Java container. In another terminal window type 'acsStartContainer –java frodoContainer'. Notice the *–java*, as you can probably guess if you wanted a C++ container you would use the flag –cpp or –py for Python. The acsStartContainer script remains active until the container itself is requested to terminate.

Once those services are up and running, type 'objexp' in another terminal. After a few seconds a GUI will pop up. On the top left-hand side of the GUI you will see a list of components. Each of these has a entry in the Components.xml file in your CDB. The one we are concerned with is a simple Java component which is described in the 'ACS Java Component Programming Tutorial'[5].
Expand 'HelloDemo' then click on 'HELLODEMO1'. If you get an error pop up then you have not installed ACS correctly. If you see a couple of operations and attributes show up in the center and right-hand areas you have installed ACS correctly!

The second way is to start up the ACS Command Center. The Command Center is a GUI and is described in detail in the ALMA Common Software Overview[7].

## 5 Programming Tutorials

ACS has many features available. To know what is available browse the web section for your ACS release. The release notes are also a very good way to know what your ACS release offers and what changes were made.

Most features of ACS have a tutorial. Although some of these tutorials are embedded into the design document of each feature. The best way to familiarize yourself with how the ACS feature works is to go through the tutorial as best you can with their examples and then create your own example (not necessarily from your subsystem).

A couple of the most useful tutorials for developers are the ones which describe how to write a component:

- ACS Java Component Programming Tutorial[5] for creating Java Components. This is the first tutorial you should do after installing ACS.
- ACS C++ Component/Container Framework Tutorial[8].
- ALMA Common Software and Python[9].

A few other tutorials which are quite helpful include:

- Notification Channel Design & Tutorial[6], though currently it is embedded with Notification Channel design. Almost every subsystem uses the Notification Channel.
- ACS Error System[10]. Also embedded with its design. The tutorials for all 3 languages are located at the end of the document.

Other documents with useful information:

- ACS Supported BACI Types[11].

# 6 API's

There is an API for the programming language available to ACS. You will also need to be familiar with the API for the IDL files since these are all the interfaces you will have access to in any language.

Java API:
http://www.eso.org/~gchiozzi/AlmaAcs/Releases/ACS_3_1/Docs/ACS_docs/java/index.html
C++ API:
http://www.eso.org/~gchiozzi/AlmaAcs/Releases/ACS_3_1/Docs/ACS_docs/cpp/index.html
Python API:
http://www.eso.org/~gchiozzi/AlmaAcs/Releases/ACS_3_1/Docs/ACS_docs/py/Acspy.html
IDL API:
http://www.eso.org/~gchiozzi/AlmaAcs/Releases/ACS_3_1/Docs/ACS_docs/idl/index.html

# 7 CDB and ACS Properties/Characteristics

## 7.1 Introduction

This section will be mainly for the benefit of C++ programmers but since ACS properties are defined using IDL and XML, Java and Python programmers could also use them.

## 7.2 What are properties/characteristics?

In any of your components you might want a particular static field defined, for example a status field. In IDL these are defined as attributes. ACS refers to these as a property or a characteristic and allows developers to create a *CharacteristicComponent* to use them. In the document ACS Basic Control Interface Specifications [14], in Section 2.1.2 and Section 3.5, you will find more details on its specifications. In the C++ programming tutorial [8] it shows you how to create a *CharacteristicComponent*.

## 7.3 Now… the tricky part… Getting it to work with your CDB!

There are three additions/modifications you need to make to some files in your CDB and/or files that are in the path that your CDB searches.

The first one that needs to be created is an XML schema which describes the properties in your IDL interface. This schema should be thought of as a type definition of your property. As mentioned in Section 4.4 your Makefile has a target CDB_SCHEMA. In your working directory you should have a directory called config and by creating the directory structure config/CDB/schema your Makefile will install into your $INTROOT any schema you have specified with the above target that is located in the above directory. It is preferable to put your schema here because it needs to be accessible to anyone who installs your software.

Next you need to tell the Components.xml file about the component which your property is in (this is basically just adding another entry for the component which uses the properties).

The second file you need to create will be an XML file that describes one instance of your property. To do this you will need to know the name of the component

specified in the Components.xml file. Then in $ACS_CDB/CDB/alma/ a directory with the name of your component will need to be created which includes the instance xml file which is called the same thing as the directory. For example,

Components.xml entry:

```
<_ Name="CPP_IMPL"
   Code="CppComponentImpl"
   Type="IDL:alma/cppexample/CppComponent:1.0"
   Container="bilboContainer"/>
```

Directory to be created:

$ACS_CDB/CDB/alma/**CPP_IMPL**

File created:

$ACS_CDB/CDB/alma/**CPP_IMPL/CPP_IMPL.xml**

If any of these three files are not present or not properly formed, your Characteristic Component will not load. The ACS C++ Component/Container Framework tutorial [8] describes these three required files in Section 3.

## 8 Dynamic Components – What's the best way to use them?

If your subsystem has decided to use dynamic components you should keep in mind there are a couple of ways of using them.

This is the preferred way to use them in Java:

```
String name = new String("ArrayController" + count);
String idl = new String("IDL:alma/Control/ArrayController:1.0");
ComponentQueryDescriptor cs =
                new ComponentQueryDescriptor(name, idl);

try {
   arrayCont = alma.Control.ArrayControllerHelper.narrow
     (m_containerServices.getDynamicComponent(cs, false));
}
catch (Exception e) {
```

```
    m_logger.severe("Failed to obtain ArrayController COB.");
}
```

The preferred way in C++:

```
ComponentSpec_var cSpec = new ComponentSpec();
cSpec->component_name = CORBA::string_dup("ArrayController");
cSpec->component_type =  CORBA::string_dup(
                    "IDL:alma/ControlArrayController:1.0");
cSpec->component_code =  CORBA::string_dup(
                                COMPONENT_SPEC_ANY);
cSpec->container_name =
          CORBA::string_dup(COMPONENT_SPEC_ANY);
ComponentInfo_var cInfo  =
                  client.manager()->get_dynamic_component(
                                              client.handle(), cSpec.in(), false);
```

The not so favorable way is to use the ACS class *ComponentSpec*. For this class you need to specify the container name that you want your dynamic component to use. By hard-coding the name of your container into your class it takes away from the dynamic-ness of dynamic components.

For Python, look at the ContainerServices class in the online API.

## 9 Programming FAQs about using ACS

As you will find out, ACS uses many outside tools and technologies. As a user of ACS you'll find out that the intersection of these elements put limitations on the well known standards of each of these tools and technologies. Basically, something that you know to be perfectly legal in one might not work in the ACS environment because of a conflict with another system.
Some of these known limitations have been documented in a FAQ section. These FAQ pages contain a lot of useful information. Familiarizing your self with these issues before hand can save a lot of time.

You can find the FAQs at the following location:
http://almasw.hq.eso.org/almasw/bin/view/ACS/AcsFAQ

## 10 Good things to know!

- As a Java programmer you must know about packages. When setting up your development environment for ALMA if you do not have your directory structure matching your Java package structure you will run into problems when you try to compile because of how the ACS Makefiles work.

    - For example: If your subsystem name is SCHEDULING and you want to define 2 packages, alma.scheduling.master_scheduler and alma.scheduling.scheduler, then your directory structure should look like this:

        SCHEDULING/src/
        SCHEDULING/src/alma/
        SCHEDULING/src/alma/scheduling
        SCHEDULING/src/alma/scheduling/master_scheduler
        SCHEDULING/src/alma/scheduling/scheduler

- If you read the ACS Java Component Programming Tutorial[5] you know about the COMPONENT_HELPERS=on flag for the Makefile. The helper file is generated from the IDL file where your interface is described. There are a couple of points to mention on the helper files.

  The first is that if there is something in your file that doesn't let the ACS IDL compiler finish with an error free compilation the component helper will not be generated. In previous versions the whole build would pass but when you watch the output of the build you will see that there were errors with the ACS IDL compiler. This is a bug and should be reported to the ACS developers.

The second issue is about the fact that there are 2 helper files generated. One is specific to CORBA and you don't need to worry about. The second is the one generated for your component. This one you do have to worry about. Many people have made the mistake of referencing the CORBA helper file when they edit their CDB's Component.xml file. Everything works until you try to instantiate your component. You will get an error message saying your helper does not implement ComponentHelper. Make sure the package name of your helper matches your implementation file.

- If using properties in your IDL you must have an instance xml file even if you define defaults in your schema.

## 11 Conclusion

It takes a bit of work to become comfortable using ACS. Like any piece of software there will be things you love about it and things you don't like so much. But once you get a good handle on how to work with it, it becomes second nature. Just remember, like every other subsystem in ALMA, ACS is also a work in progress!