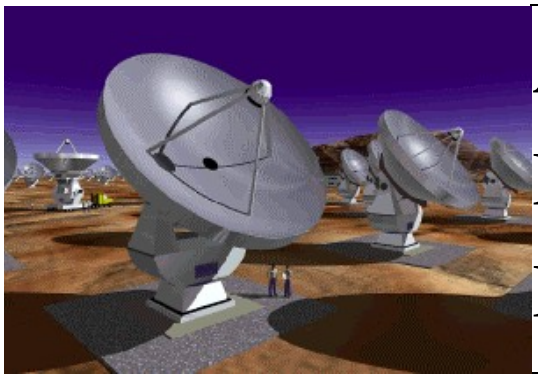

Atacama Large Millimeter	ALMA-xxxx Issue: 3.1.0-pre 2004-03-02

ALMA Common Software
Overview

Gianluca CHIOZZI
European Southern Observatory
Matej Šekoranja
Jozef Stefan Institute

Keywords: ACS, Installation, Overview

Author Signature:	Date:
Approved by: Institute:	Signature: Date:
Released by: Institute:	Signature: Date:

Change Record

REVISION	DATE	AUTHOR	SECTIONS/PAGES AFFECTED
	REMARKS		
1.0	2001-09-24	G.Chiozzi	All.
	Document created from notes prepared by M. Šekoranja		
1.0-Rev.1	2001-10-08	G.Chiozzi	Added
	Added section on Environment Variables. Added minor details and fixed typos.		
1.0-Rev.2	2001-10-23	G.Chiozzi	First page
	Replaced Advanced Common Software with ALMA Common Software		
	2001-11-13	G.Chiozzi	Sec. 2 and 3
	Fixed a few typos.		
1.0-Rev.3	2001-11-22	B. Jeram	Sec. 2.3.8 and 2.3.9
	Added information about ORB ports for maciActivator and maciManager		
1.1	2002-04-05	G.Chiozzi	All
	Modified for ACS 1.1		
1.1-Rev.1	2002-06-28	G.Chiozzi	All
	Updated with suggestions/feedback after first set of installations		
2.0	2002-12-06	G.Chiozzi	All
	Updated for ACS 2.0		
2.1	2003-06-09	G.Chiozzi	All
	Updated for ACS 2.1		
2.1.1	2003-07-07	G.Chiozzi	All
	Updated environment variables section		
3.0.0	2003-11-15	G.Chiozzi	All
	Updated for ACS 3.0		
3.0.0.1	2003-12-29	G.Chiozzi	All
	Updated for ACS 3.0. Fixed errors.		
3.0.1.0	2004-02-18	G.Chiozzi	All
	Updated for ACS 3.0.1 and according to comments from J.Schwarz. Added captions. Removed Alarm Display. Added instructions to start LCU Containers.		
3.1.0	2004-05-11	M.Schilling	Sec. 2.2, 2.3, 3.2
	Now mentions acsList, and general “–help” option. Adminc now outdated.		
5.0.4	2006-07-24	G.Chiozzi	Sec. 6
	Updated ports usage information.		

Table of Contents

1 Summary....5

Glossary....5
References....5

2 Overview of ALMA Common Software 3.1....6

2.1 ACS Command Center....7
2.2 ACS Instance....7
2.3 ACS Startup....8
2.4 ACS Shutdown....9
2.5 ACS Processes....9
2.5.1 acsStartContainer....9
2.5.2 acsStopContainer....10
2.5.3 acsStartManager....10
2.5.4 acsStopManager....11
2.5.5 acsStartJava....11
2.5.6 acsLogSvc....12
2.5.7 cdbjDAL....12
2.5.8 CORBA Services....14
2.5.9 loggingClient....15
2.5.10 loggingService....16

3 ACS User Interface utilities....18

3.1 ACS Command Center....18
3.2 Administrator Client....18
3.3 CDB Browser....18
3.4 Logging User Interface....19
3.5 Object Explorer....20
Event Browser....22

4 Structure of the Configuration Database....23

Resolving the Configuration Database Reference....23
Database configuration files....23
Manager Configuration Database....23
Manager's own CDB branch....23
Component definitions for Manager....24
Container Configuration Database....25
Container's own CDB branch....25

Component's definitions for Manager.....25
Characteristic Component Configuration Database.....25
Configuration Database.....26

5 ACS Environment Variables.....26

5.1 Most Important environment variables.....26
5.2 Other environment variables set in .bash_profile.acs and used by ACS:.....28
5.3 Environment variables deprecated or not used any more:.....31

6 TCP Ports Allocation for ACS 32

1 Summary

This is a simple Overview of the ALMA Common Software (ACS) version 3.1. Read it to get an idea of what is provided and how to get the system up and running. It is also useful to read this manual if you have been using ACS 3.1 and are upgrading to 3.1.

This document is not intended to be an introduction to ACS features and concepts.

For details on ACS concepts, architecture and APIs look at the main ACS Web Page1 or directly at the index page of the ACS 3.1 Online Documentation1 and pick the documents you are most interested in. The ACS Web Page also contains some introductory papers presented at conferences that provide an overview of ACS concepts.

Glossary

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

References

All documents referenced here are available in the online ACS 3.1 documentation

[1] ACS Web Page: (http://www.eso.org/projects/alma/develop/acs)

[2] ACS 3.1 Online Documentation: (http://www.eso.org/projects/alma/develop/acs/Releases/ACS_4_0_Docs/index.html)

[3] ACS Installation Manual - v. 4.0

[4] ACS Configuration Database, CDB

[5] VLT Common Software - Installation Manual - VLT-MAN-ESO-17200-0642 v.1.14

[6] VLT Software - Tools for Automated Testing User Manual - VLT-MAN-ESO-17200-0908 v.1.5

2 Overview of ALMA Common Software 3.1

The following component diagram shows the processes running in a typical ACS 3.1 session and the main relationships between them.

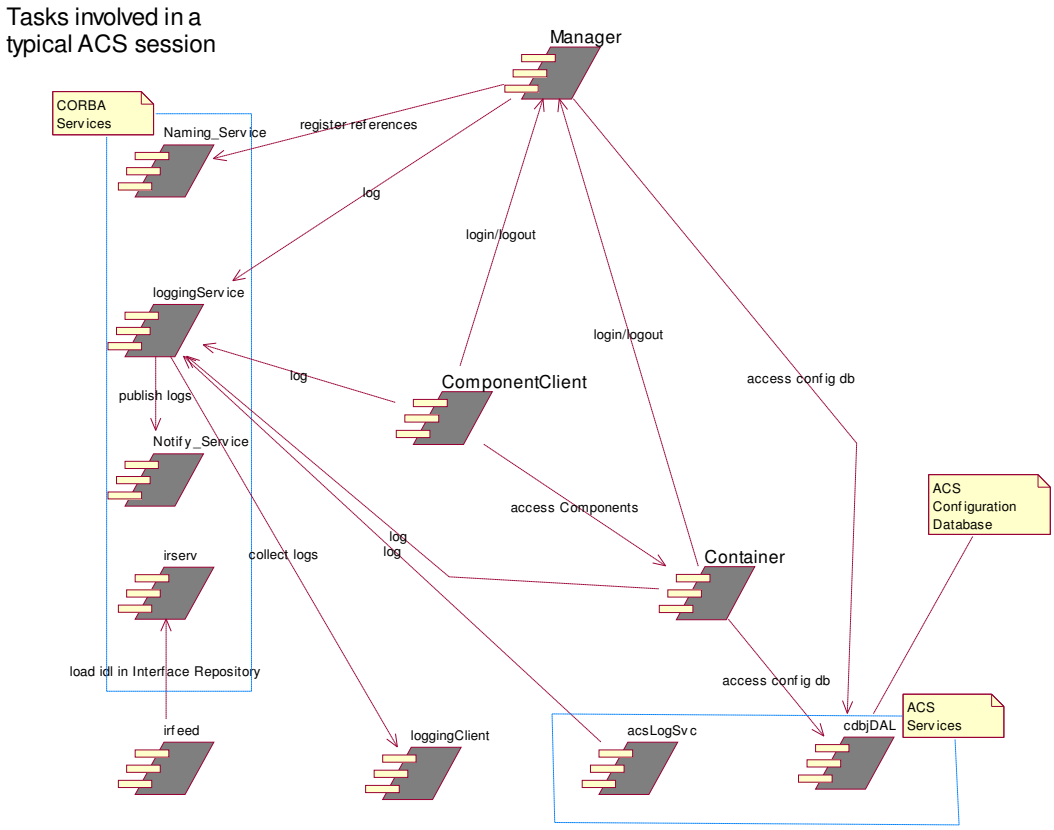

Figure 1- ACS Tasks Diagram

The following sections will provide some introductory details on the functionality covered by these processes and ACS startup/shutdown procedures.

In general, we assume that there will be one `ACS Manager` and all `CORBA services` running on the main Linux workstation. As of ACS 4.0, it is also now possible to federate different instances of ACS via `ACS Manager` federation. There

will be one C++ Container per LCU and possibly other Container processes on multiple workstations. Java Containers and Clients can run anywhere including MS-Window PCs.

2.1 ACS Command Center

ACS can be started, shutdown and administered from the ACS Command Center GUI.

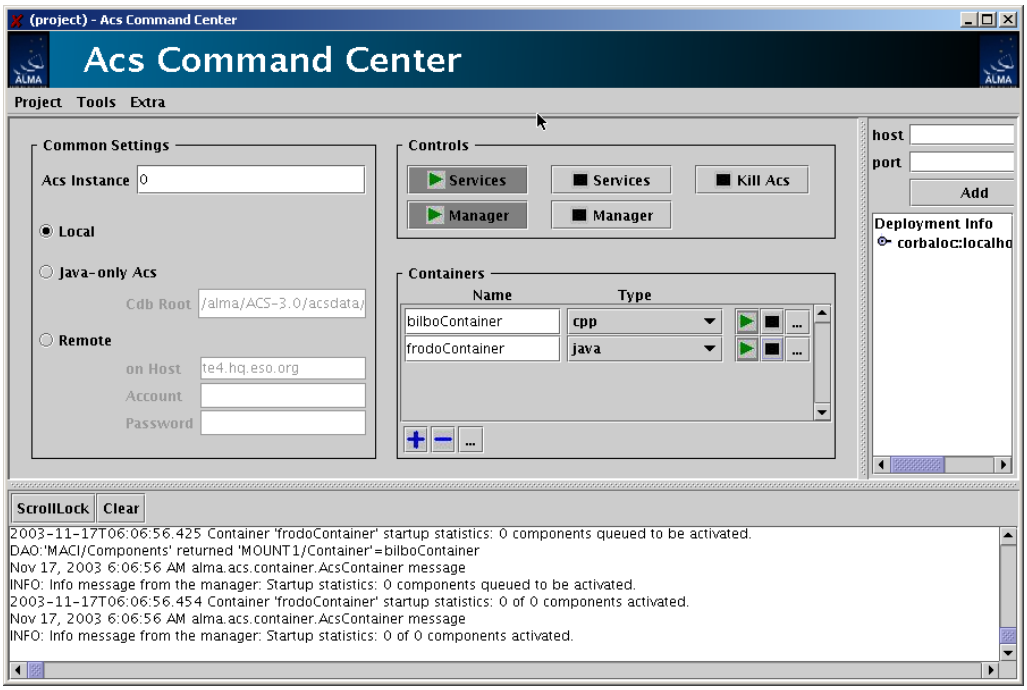

Figure 2- ACS Command Center

Through the Command Center you can start/stop ACS services, Manager and Containers as well as ACS tools like the Object Explorer. Your preferred configuration can even be saved and reused.

To start the application type the following command:

```
> acscommandcenter
```

2.2 ACS Instance

In order to allow multiple users to work in completely separate sandboxes on the same machine, we have introduced the concept of “ACS Instance”

The environment variable ACS_INSTANCE can assume the values from 0 (default) to 9.

Based on this environment variable (and/or the –b <ACS_instance #> option for most ACS commands), ports used by ACS services and processes are automatically calculated according to the rules described in section 6.

Users selecting two different values for ACS_INSTANCE will work without interfering with one another. Which instances are currently used on a machine can be determined with the `acsList` command.

2.3 ACS Startup

Command line commands are available to start and stop ACS processes. Invoking most of the commands with `--help` will provide usage information.

Before Containers can be started, ACS services and Manager must be active. Therefore the following commands must be executed:

1. CORBA Services and Manager:
   `> acsStart`
   `This includes` also the IDL server, logging (acsLogSvc), and the Configuration Database (cdbjDAL).
   This command returns control when the services and Manager are up and running.

2. any needed Container:
   `> acsStartContainer <-cpp|-java|-py> <Container name>`

3. any clients…

We suggest running each command in an independent window/xterm to be able to look at the output produced.

The command `acsStart` will try to start everything according to the value of the `ACS_INSTANCE` variable (0 by default). Since there can be only one ACS instance running for a given value of ACS_INSTANCE, if `acsStart` finds out that ACS processes are already running with the selected value, it will automatically select a new one and print the choice to standard output.

Both acsStart and acsStartContainer, like most other ACS commands, accept the command line option:

-b <acs instance #>

This overrides the value of the `ACS_INSTANCE` environment variable.

It should be noted that the time allowed for most of the ACS startup commands to complete can be increased by increasing the value of the `ACS_STARTUP_TIMEOUT_MULTIPLIER` environment variable (an integer greater than 0). The same behavior can be duplicated by using the following command-line option available in most startup and shutdown scripts:

-t <integer timeout multiplier value>

This overrides the value of `ACS_STARTUP_TIMEOUT_MULTIPLIER`.

2.4 ACS Shutdown

ACS processes can be stopped with the following command:

- `acsStop` [-b <acs instance #>]

The command:

- `killACS [-Q]`

will instead try to **KILL** all ACS processes in the host for all `ACS_INSTANCE` values and clean things up. The optional –Q parameter will kill ACS processes much quicker but it will not kill all ACS clients. It is an (almost) last resort option to cleanup a host from dangling ACS processes. In very rare cases `killACS` might not be sufficient. The *real* last resort is to `kill -9` the dangling processes/threads.

2.5 ACS Processes

2.5.1 acsStartContainer

The *acsStartContainer* command is used to start ACS containers for C++, Java or Python. To successfully resolve object interfaces (IDL IDs), C++ containers use the Interface Repository Service. In order to use BACI recovery, Container must use a dedicated ORB port (by default 3050 for C++, 3052 for Java, 3054 for Python and ACS_INSTANCE=0).

The ORB port can be changed using the command line option `--port` or (for C++ Containers) the same option in the configuration database entry named `Command Line`:

```
acsStartContainer [ --port < 0-25 | precise port number > ]
                  < -cpp | -java | -py > <Container name>
```

If the Container is running on the same host as the ACS Manager and there are no other Containers running on the host, it is not necessary to pass any configuration parameters:

```
acsStartContainer < -cpp | -java | -py > <Container name>
```

If the Container runs on another host, the corbaloc for the reference to the manager shall be given on the command line or in the configuration database:

```
acsStartContainer
           -m corbaloc::<manager_host>:<manager_port>/Manager
           < -cpp | -java | -py >  <Container name>
```

Manager reference is resolved using algorithm described in 2.5.3.1.

For information on the Container's CDB data see section 4.

Other command-line options can be examined using the –h option.

Instructions on starting a Container under a VxWorks LCU are available in the following ACS FAQ:

http://almasw.hq.eso.org/almasw/bin/view/ACS/FAQVxWorksContainerStart

2.5.2 acsStopContainer

acsStopContainer is used to shut down one or more Containers whose reference is retrieved by name from the Manager. Wildcards can be used to specify the name of the Container(s) to be shutdown. To resolve Manager's reference, the algorithm described in section 2.5.3.1 is used:

```
acsStopContainer <Container name(s)>
   [-m corbaloc::<computer address>:<manager_port>/Manager]
```

Other command-line options can be examined using the –h option.

2.5.3 acsStartManager

ACS provides a Java Manager.

To start the Manager, run the command:

- acsStartManager

By default the Manager ties up TCP port 3000 (for `ACS_INSTANCE=0`). This can be changed by using the command-line option `-ORBEndpoint` (also settable in the configuration database's `Command Line` entry):

```
-ORBEndpoint iiop://<host>:port
```

If `-ORBEndpoint` option is not specified from the command-line or in the configuration database and the environment variable MANAGER_REFERENCE is defined, then the port number is extracted from it. The environment variable has to be of the form:

```
corbaloc::<manager_host>:<port>/<Manager_name>
```

Manager uses the Naming Service to map activated Components and some special references *(NameService, InterfaceRepository, Log, LogFactory, NotifyEventChannelFactory, ArchivingChannel, LoggingChannel).* Since Manager is the

central communication point, it is necessary that all applications retrieve the needed references through the Manager and not directly through the NameService.

```
acsStartManager [-ORBInitRef
NameService=corbaloc::<ns_host>:
          4000/NameService]
```

To minimize system configuration, Manager uses the following algorithm to resolve *NameService* references:

1. Command-line corbaloc
2. CDB option `-ORBInitRef`
3. Environment variable `NAMESERVICE_REFERENCE`
4. Using generated reference:
   `corbaloc::<hostname>:<ns_port>/NameService`

For information on the Manager's CDB data see section 4. To see other command-line options for Manager, use the –h option.

2.5.3.1 Resolving Manager reference algorithm

ACS services and clients use the following algorithm to resolve Manager reference via the API function `maci::MACIHelper::resolveManager()`:

1. Command line option: `-m` or `–managerReference`
2. ManagerReference value in the configuration database. This is only applicable to C++ Containers.
3. Environment variable MANAGER_REFERENCE
4. Using generated reference: corbaloc::<hostname>:<manager_port>/Manager

2.5.4 acsStopManager

acsStopManager is used to remotely shut down the Manager. To resolve Manager's reference, the algorithm described in section 2.5.3.1 is used:

```
acsStopManager

   [-m corbaloc::<computer address>:<manager_port>/Manager]
```

Other command-line options can be examined using the –h option.

2.5.5 acsStartJava

This script is used on Linux to execute any ACS Java application.
It automatically builds the $CLASSPATH environment variable locating all needed jar files and of setting all Java properties used by ACS/ABeans Java applications:

```
acsStartJava <java_class_name>
```

For example:

```
acsStartJava alma.acsabeans.examples.PSPanel.PSPanel
```

Other command-line options can be examined using the –h option.

2.5.6 acsLogSvc

This ACS process activates a logging IDL interface implementation that any CORBA-aware client can use to generate Log messages of the types specified in the ACS Logging and Archiving specifications without having to manually format the corresponding XML string. The acsStartORBSRVC script automatically executes this command.

2.5.7 cdbjDAL

This process provides the IDL DAL (Data Access Layer) interface to the ACS xml-based configuration database (CDB).

For more details, see section 4 and the Configuration Database manual 1.

2.5.7.1 cdbjDAL command

This command starts the Configuration Database service. The `acsStartORBSRVC` script automatically executes it.

The Configuration Database service uses the following algorithm to resolve the path of the directory where the CDB XML files are located:

1. Command line option `-root <path>`
2. Environment variable `ACS_CDB`

Once the path has been resolved, it expects to find CDB XML files in the CDB sub-directory.

When `cdbjDAL` is executed from within `acsStartORBSRVC` it is not possible to pass the `-root` option. Therefore the only real usable way of defining the path for the CDB files is using the environment variable `ACS_CDB`. The default value for this variable provided by the login scripts is `$ACSDATA/defaultCDB`, where the ACS installation procedure puts the standard sample CDB.

At run-time, the `cdbjDAL` service searches for schema definitions, in the given order, in the following directories:

```
CDBPATH= $ACS_CDB/CDB/schemas
```

```
$PWD/../config/CDB/schemas:
$INTROOT/config/CDB/schemas:
$ACSROOT/config/CDB/schemas
```

Other command line options:

-o <file> writes in the given file the IOR

-OAport specify a TCP port different from the default

2.5.7.2 cdbjDALClearCache command

This command requests the Configuration Database (jDAL implementation) clear its internal cache and to notify applications (in particular the Manager) that updated data may be available.

Using this command, it becomes easy to modify the CDB and get the new values “active” without having to restart ACS processes:

- Edit with a text editor jDAL CDB XML files
- Issue cdbjDALClearCache command to notify the CDB
- The Manager gets automatically notified and updated and reloads its data from the CDB at the next request.

2.5.7.3 cdbjDALShutdown command

This command shuts down the Configuration Database service. The acsStopORBSRVC scripts executes this command.

Useful command line options:

-k corbaloc::<HOST>:<CDB_PORT>/CDB
Host and port where the cdbjDAL is running

2.5.7.4 cdbRead command

This command is a utility to directly dump DAOs from the Configuration database.

The calling syntax is:

cdbRead <DAO name> [-raw]

where <DAO name> is the hierarchical name of the DAO in the CDB, for example the command:

```
cdbRead /alma/LAMP1
```

produces the output:

```
Node brightness
  description="brightness"
  units="%"
  min_step="1.0"
  min_value="0.0"
  max_value="100"
  default_timer_trig="10000000"
  min_timer_trig="10000"
  min_delta_trig="0"
  default_value="0.0"
  graph_min="-1.7976931348623157E+308"
  graph_max="1.7976931348623157E+308"
  archive_delta="0"
  format="%9.4f"
  resolution="65535"
  archive_priority="3"
  archive_min_int="0"
  archive_max_int="0"
```

Other command line options:

-raw as last parameter after the DAO name, dumps the DAO as a raw XML file.

-k corbaloc::<HOST>:<CDB_PORT>/CDB
Host and port where the cdbjDAL is running

2.5.8 CORBA Services

ACS uses a number of standard CORBA Services, which need to be started before any ACS specific process.

The scripts
`acsStart` (or, more precisely, `acsStartORBSRVC`)
and
`acsStop` (or, more precisely, `acsStopORBSRVC`)
are used to start one by one the CORBA Services, passing the necessary command-line parameters. All Services are started on the local host and it is assumed that also the ACS Manager process will be started on the same host.

In particular, it is necessary to configure each service to run on a specific TCP port (see chapter 3) and to pass the `-ORBDottedDecimalAddresses=1` option to allow communication with LCUs.

These scripts also start/stop the Logging Service (see 2.5.10), acsLogSvc and cdbjDAL server.

2.5.8.1 Naming Service (TAO)

```
Naming_Service –ORBEndpoint iiop://<host>:<ns_port>
               –ORBDottedDecimalAddresses=1
```

2.5.8.2 Notify Service (TAO)

TAO Notify Service requires Naming Service to run.

```
Notify_Service –ORBInitRef
    NameService=corbaloc::<ns_host>:<ns_port>/NameService
    –ORBEndpoint iiop://<host>:<notify_port>
    –ORBDottedDecimalAddresses=1
```

```
Where <ns_host> is the host where the Names Service is
running.
```

2.5.8.3 Interface Repository (MICO)

MICO Interface Repository does not automatically register its reference to the Naming Service. This has to be done via script and the TAO *nsadd* tool ($ACS_ROOT/TAO/utils/nslist/nsadd):

```
$MICO_HOME/bin/ird –ORBNoResolve –ORBIIOPAddr inet:$HOST:
$ACS_IR_PORT --ior $IR_IOR &
```

```
$ACE_ROOT/TAO/utils/nslist/nsadd --name InterfaceRepository
--ior `cat $IR_IOR` –ORBInitRef NameService=corbaloc::
$HOST:$ACS_NAMING_SERVICE_PORT/NameService $ORBOPTS
```

2.5.8.3.1 Feeding the Interface Repository

The interface repository needs to be given all ACS IDL interfaces. The generic command syntax is:

```
acsIrfeed  [–IRcorbaloc
    corbaloc::$HOST:$ACS_IR_PORT/InterfaceRepository]
```

The `acsIrfeed` command can also be used at any time (and in particular during development and debugging) to (re-)load speficic IDL interfaces in the Interface Repository:

```
acsIrfeed  <filename>.idl
```

2.5.9 loggingClient

Logging client is a simple example of an application that attaches to the Notification Service, retrieves ACS Logs or Archive Monitors, and displays them on standard output. It is a structured push event consumer. For its operation the Naming Service is queried to

resolve references of notify channels. The current implementation has the capability of monitoring the Logging and Archiving channels.

```
loggingClient LoggingChannel
   –ORBInitRef NameService=corbaloc::<ns_host>:
        <ns_port>/NameService
```

or

```
loggingClient ArchivingChannel
    –ORBInitRef NameService=corbaloc::<ns_host>:
        <ns_port>/NameService
```

It is often convenient to use "`| tee fileName`" to store at the same time the logs in a file.

2.5.10 loggingService

This implements ACS 3.1 Centralized Logger (Telecom Logging Service – see Logging and Archiving specification). *loggingService* activates Telecom Logging Service *Log* and *LogFactory* interfaces and creates *ArchivingChannel* and *LoggingChannel (with domain_name=Logging)*. All these objects are registered with the Naming Service and are accessible via Manager using get_service() method.

loggingService requires Naming Service and Notify Service to run.

Running *loggingService*:

```
loggingService
   –ORBInitRef NameService=corbaloc::<ns_host>:
            <ns_port>/NameService
   –ORBDottedDecimalAddresses=1
   –ORBInitRef     NotifyEventChannelFactory=corbaloc::
       <notify_host>:<notify_port>/
   NotifyEventChannelFactory
```

where <notify_host> is the host where the Notification Service Event Channel Factory is running.

Specifying -ORBInitRef NotifyEventChannelFactory option is optional. NotifyEventChannelFactory reference is obtained from the Name Service, but -ORBInitRef has higher priority then NS lookup, algorithm:

1. resolve_initial_references
2. Name Service lookup

For more detailed information about logging and archiving, see the Logging and Archiving specification.

Notice that logs are published on the Notification Channel as XML strings.

If no application (such as loggingClient) listens to the Notification Channel, the logs are lost.

2.5.10.1 Controlling logging behavior

The logging system uses as much as possible the standard Logging APIs of C++ and Java.

See the ACS Logging System documents for details.

C++ Logging System

C++ Logging system is based on ACE Logging API.

Several macros are defined in `$ACSROOT/include/logging.h` to make message logging simpler for user applications. All messages which are logged using the logging macros are sent to the ACS loggingService except when the application is unable to connect to the loggingService (for example, if the loggingService is not running). In this case, log messages are sent to a local text file. Each process/thread writes its own logging text file, by appending its process name and PID to a standard root.

By default, the standard root name is:

```
$ACSDATA/tmp/acs_local_log_
```

For example, ACS Manager would write a file like:

```
$ACSDATA/tmp/acs_local_log_maciManager_12345
```

It is possible to change the default destination for ALL ACS temporary files from $ACSDATA/tmp to any other directory by setting the environment variable $ACS_TMP.

It is possible to change the root name for logging files by setting the environment variable $ACS_LOG_FILE. If $ACS_LOG_FILE is set, $ACS_TMP is ignored.

The environment variable `ACS_LOG_STDOUT` can be used to control the amount of information sent to `stdout` at a per-process level. By default, only log messages with priority equal or higher than LM_INFO are sent to `stdout`. If `ACS_LOG_STDOUT>0`, all log messages with `priority >= ACS_LOG_STDOUT` are also sent to `stdout`.

The C++ logging system caches log messages on a per-process basis before transmitting them to the centralized logging service. Several configuration database parameters control whether messages are to be logged at all, which messages should be cached locally, and which messages should be transferred immediately to the logging service (or the local log file):

- Messages with a priority less than `MinCachePriority` are not logged at all. Manager and Container provide a CDB point with this characteristic (see sections 4 and 4). By default, `MinCachePriority` is set to zero so that all messages are logged.
- Messages with a priority greater than `MinCachePriority` but less than or equal to `MaxCachePriority` are cached locally. When the cache fills up, all cached messages are transferred to the logging service.
- Messages with a priority greater than `MaxCachePriority` are logged immediately, bypassing the cache.
- The cache size for Manager and Container can be controlled in their CDB definition using the `CacheSize` characteristic (see sections 4 and 4). If `CacheSize` is set to 0 or 1, all messages (except those with priority less than `MinCachePriority`) are logged immediately without caching.

For more information, look at the online documentation for the `LoggingProxy` class.

This is the class used by applications to interact with the loggingService.

Java Logging System

The ACS Java Logging API is based on the official JSDK Java Logging java.util.logging and it has been integrated with the implementation of the CORBA Telecom Logging Service and the rest of the ACS.

3 ACS User Interface utilities

ACS provides some generic GUIs to administer and interact with a running system.

Each of these applications is described in detail in a specific user manual.

3.1 ACS Command Center

Used to startup/shutdown and administer ACS. Already described briefly in section 2.1

3.2 Administrator Client

The Administrator Client is outdated; its functionality is now provided by the ACS Command Center (see above).

3.3 CDB Browser

This application is used to browse the run-time configuration database.

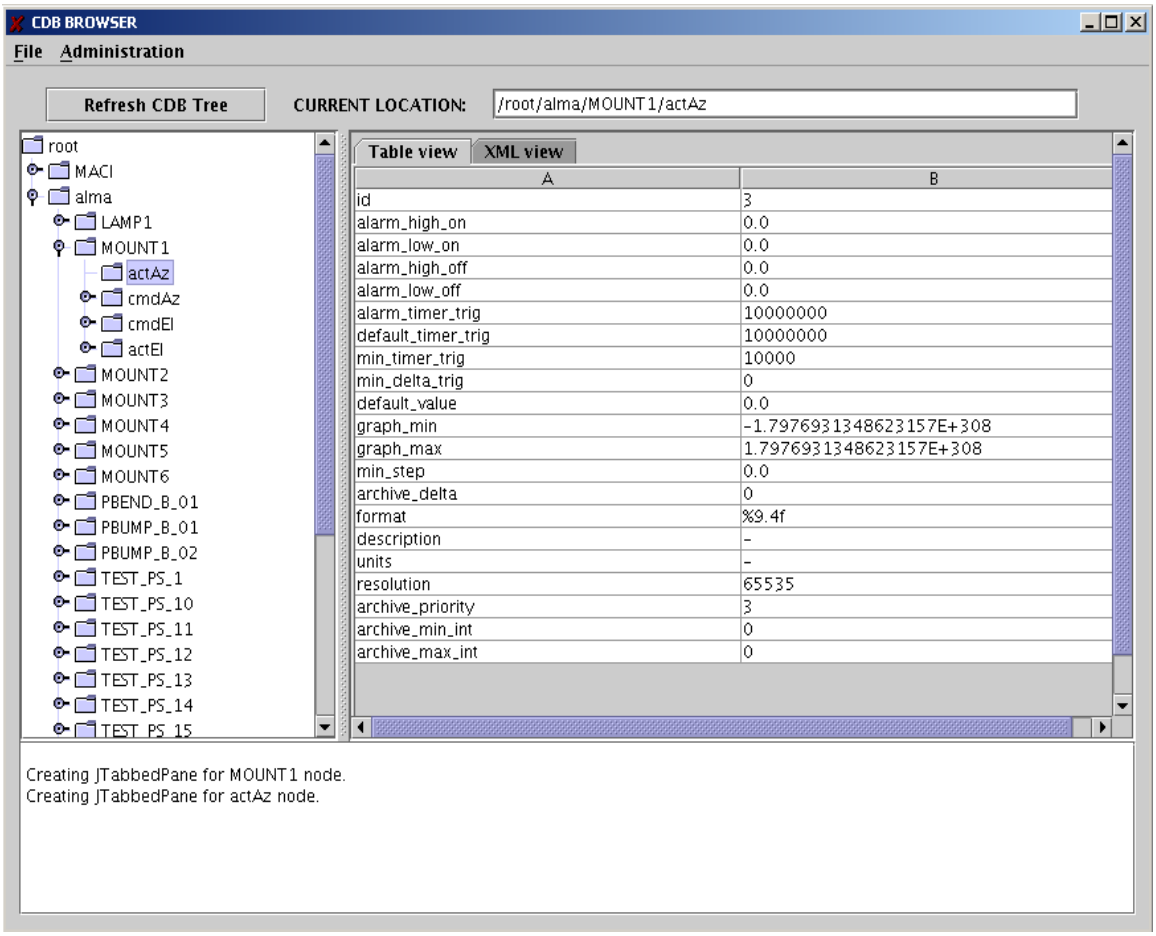

Figure 3- CDB Browser

3.4 Logging User Interface

This application is used to display logging system messages.

You can run it with the command:

> jlog

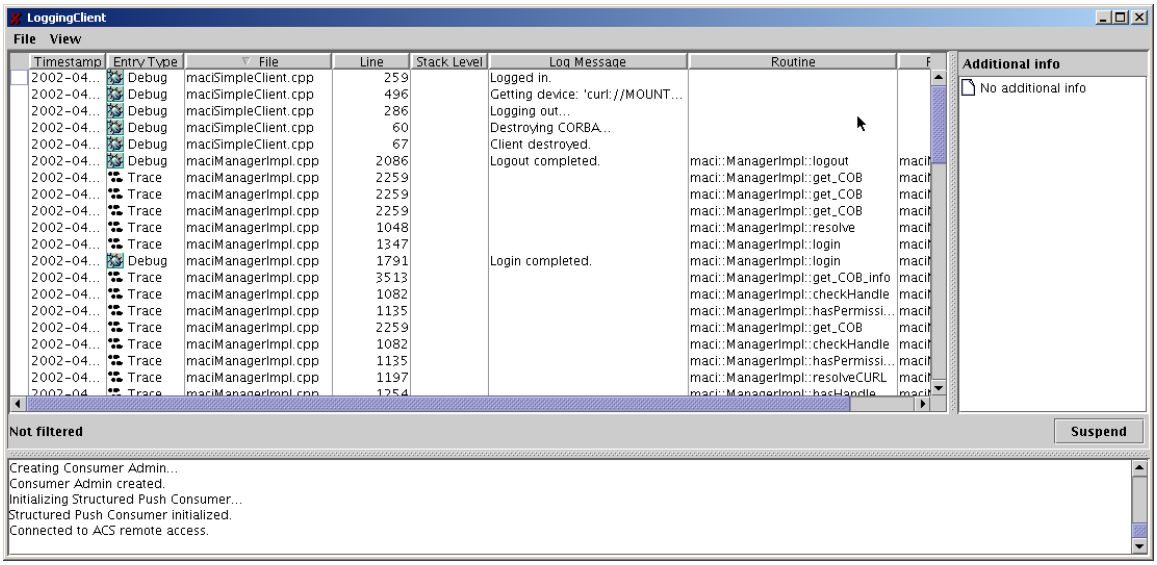

Figure 4- Logging GUI

3.5 Object Explorer

This application is used to navigate through the hierarchy of Components in the system. It allows you to call each method, get/set Properties and view Characteristics, install monitors, and draw trend plots.

You can run it with the command:

> objexp

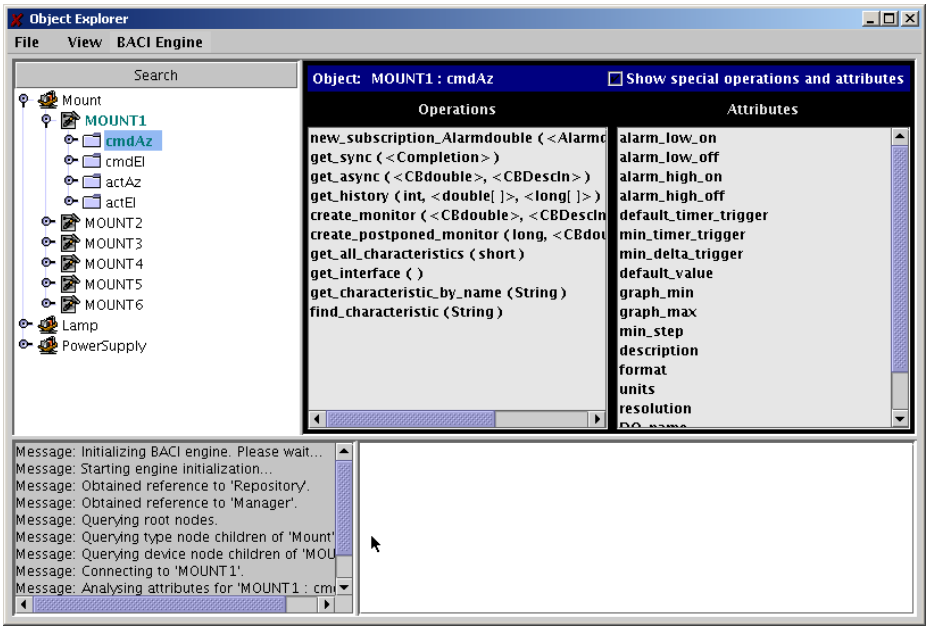

Figure 5- Object Explorer

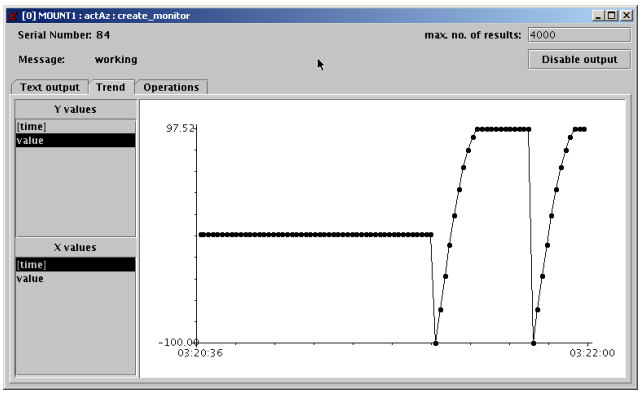

Figure 6- Property Monitor trend plot

You can pass to the `objexp` command-line all options and Java virtual machine parameters that can be passed to the `acsStartJava` command.

In particular you can pass the JVM option

```
-Dobjexp.pool_timeout=<time in ms>
```

to set DSI pool timeout, that by default is 5000ms.

For example the following command line:

```
objexp -m corbaloc::te98:3000/Manager -Dobjexp.pool_timeout=60000
```

will configure the Object Explorer to wait 60 seconds before assuming that a timeout is occurred and to connect to the Manager explicitly specified on the command-line.

Event Browser

A browser designed to monitor all events generated by the ACS event channel APIs was introduced with ACS 3.1. A few notes on this:

- The name of the executable which starts the GUI is *acseventbrowser*.
- This GUI is started automatically by clicking the *Event Browser* option in the *Tools* menu of *acscommandcenter*.

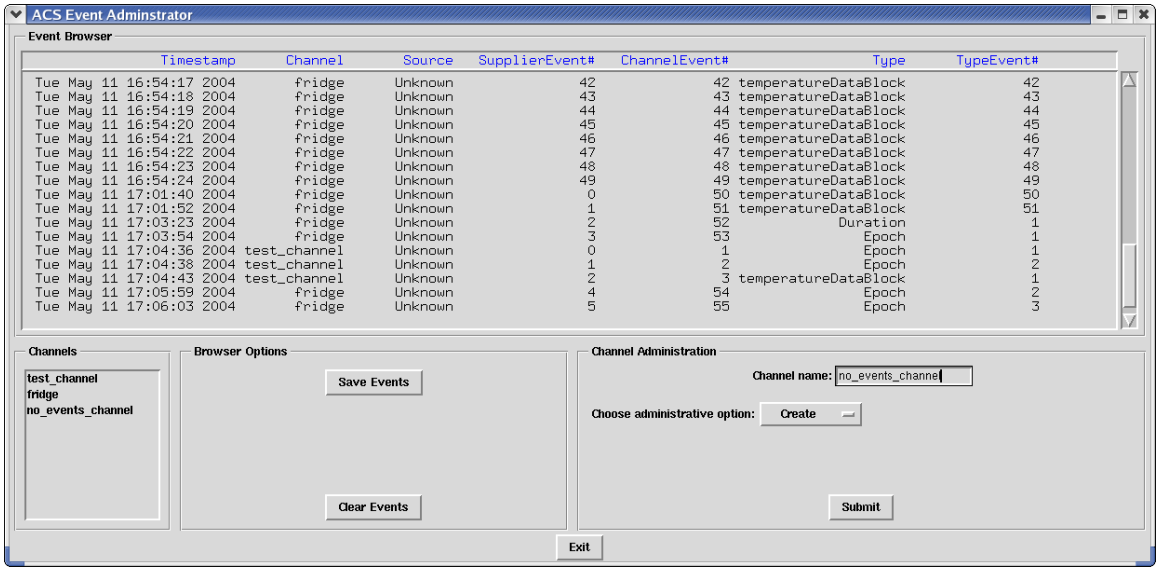

Figure 8 – Event Browser

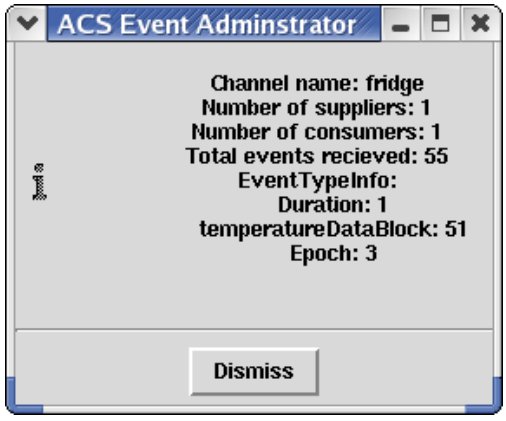

Figure 9 – Panel started by clicking the "fridge" channel in the Channels box.

4 Structure of the Configuration Database

For more information on the ACS 3.1 Configuration Database syntax, look at the ACS Configuration Database manual1.

Resolving the Configuration Database Reference

ACS applications (like acsStartManager or acsStartContainer) use the following algorithm to resolve the DAL reference:

1. Command line option –d or `-DALReference`
2. Environment variable DAL_REFERENCE
3. Using generated reference: corbaloc::<hostname>:<dal_port>/CDB

Database configuration files

Each Configuration Database is defined by a set of XML ASCII files in the directory:

```
$ACS_CDB/CDB
```

A standard configuration database contains three sub-directories:

alma – This directory contains configuration data for Components in the ALMA system, if they actually have configuration stores in the CDB. `CharacteristicComponents` ALWAYS have configuration data, but simple `Components` do not require having any data in the CDB..

MACI – This directory contains configuration data for MACI Manager, Containers and the Components' main configuration file used by ACS Manager to map Component names into their implementation and the Container responsible for them.

schemas – This directory contains schema files used to resolve default values and inheritance in CDB DAO instances. Each DAO instance in the Configuration Database is represented by an XML file and shall be an instance of an XML Schema file in the `schemas` directory or in the config/CDB/schemas sub-directory of $INTROOT or $ACSROOT. When the cdbjDAL server gets a request for a DAO, the XML parser uses the corresponding XML Schemas to expand the complete structure of the DAO.

Manager Configuration Database

Manager's own CDB branch

The Configuration Database for the Manager is in the database branch:

```
/MACI/Managers/Manager
```

The definition of the configuration parameters for the Manager is in schema file:

```
$ACSROOT/config/CDB/schemas/Manager.xsd
```

Important parameters are:

```
CommandLine: Default command-line added to given command-line
Startup: List of Components to be automatically created on
startup

/// Logging configuration data
CacheSize          25      number of logs to be cached before
                           logging
MinCachePriority    0      minimum log priority
                           messages with lower priority
                           are ignored
MaxCachePriority   31      maximum log cache priority
                           messages with higher priority are not
                           cached and are logged immediately
```

Component definitions for Manager

For each Component in the system, the Manager must be able to find on request all information needed to return its reference to clients and to start/stop it when commanded.

Therefore, the Configuration Database used by the Manager MUST also contain the file:

```
/MACI/Components/Components.xml
```

The definition of the Components configuration file is in the schema file:

```
$ACSROOT/config/CDB/schemas/Components.xsd
```

This consists of a table with one record per each known Component, like in the following example:

```
<_ Name="PBEND_B_01" Code="acsexmplPS"
                    Type="IDL:ALMA/PS/PowerSupply:1.0"
                    Container="Container"/>
```

- Three attributes must be defined for each of these instances:

```
/// Executable for the Component
/// i.e. Component`s DLL name ('xxx'
/// on Unix becomes 'libxxx.so')
/// for C++ Container Components (or Java class for Java
/// Components)
Code         for example "ps"

/// Component`s type name, i.e. its IDL interface
Type         for example "IDL:ALMA/PS/PowerSupply:1.0"

/// Name for the Container where the
```

```
/// Component shall be activated
Container     for example "abm1”
```

Changing this information allows relocating Components and replace versions without having to stop and recompile the system.

Container Configuration Database

Container's own CDB branch

The Configuration Database for a C++ Container is in the database branch:

```
/MACI/Containers/<Container name>
```

The CDB entry for a C++ Container is optional (defaults are used if not present), while Java Containers currently do not support storing configuration parameters in the CDB. Python Containers have only limited support for Container info at the present.

The definition of the configuration parameters for the C++ Container is in schema file:

```
$ACSROOT/config/CDB/schemas/Container.xsd
```

Important parameters are:

```
CommandLine: Default command-line added to given command-line
ManagerReference       Manager reference in the format
                            corbaloc::<host>:<port>/Manager
Autoload: DLLs to be loaded automatically on Container startup

/// Logging configuration data
CacheSize          25      number of logs to be cached before
                           logging
MinCachePriority    0      minimum log priority
                           messages with lower priority
                           are ignored
MaxCachePriority   31      maximum log cache priority
                           messages with higher priority are not
                           cached and are logged immediately
```

Component’s definitions for Manager

Whenever the Manager requests a Container for a Component, it passes to it information about the DLL to be loaded.

Characteristic Component Configuration Database

Characteristic Components, i.e. Components implemented according the BACI Design Patterns, keep the configuration for their Properties and Characteristics in the CDB.

Each Characteristic Component looks for its configuration information in the database branch:

```
/alma/<Component name>
```

The actual structure of the database depends on the type of Component, but will essentially contain characteristics for each Property as defined in `$ACSROOT/config/CDB/schemas/BACI.xsd`

Configuration Database

After the installation of ACS 3.1, the directory

```
$ACSDATA/config/defaultCDB:
```

will contain a sample configuration database.

This is the configuration database that is used by default when ACS is started.

This database declares a number of Component instances that are defined in the ACS example module `acsexmpl` as well as the Java and Python example modules (jcontexmpl and acspyexmpl).

These Components are provided as examples for the users of ACS and for testing ACS.

It is suggested to make a backup copy of `defaultCDB` and to modify your own copy. It is also a good practice to create new Configuration Databases for specific applications, assigning them explicit names, like `corrCDB`, instead of using defaultCDB all the time.

But do NOT forget to export the environment variable `ACS_CDB` to point to the configuration database instance you want to use.

5 ACS Environment Variables

The configuration of ACS is determined by a number of environment variables.

Generally, these variables are set to a proper default value by the default login script whose template is available in $ACSROOT/config/.acs/.bash_profile.acs. Each user has to take care of copying this script into his home directory (typically in the .acs directory), eventually adapting it and executing it to prepare the environment.

Some variables have been already described in the previous sections, but we give below a list of the most interesting ones.

5.1 Most Important environment variables

ACS_CDB (default: $ACSDATA/config/defaultCDB)
Location of the configuration database files to be used when cdbjDAL is started.

ACSDATA (default: /alma/ACS-<version>/acsdata)

The root directory where all ACS configuration files (including Configuration Database files) are stored.

ACS_LOG_FILE ($ACSDATA/tmp/acs_local_log_, when not set)

Location and root file name for ACS log files. Each process/thread generates a unique file name by appending process name and PID to the root file name. See section 2.5.10.1.

ACS_LOG_STDOUT

Control the amount of information sent to `stdout`.
By default, only LM_INFO log messages are sent to `stdout`.
See section 2.5.10.1.

ACS_NAME_SERVICE (default: $MANAGER_COMPUTER_NAME:4000/NameService)

CORBA Initial Reference for the Naming Service.
Usually the reference of the Manage is sufficient for applications, but some special applications, like cdbjDAL need it.

ACSROOT

The place where ACS CMM modules are installed. For ACS 3.1 this is typically: /alma/ACS-3.1/ACSSW

ACS_STARTUP_TIMEOUT_MULTIPLIER

An integer value used to increase the amount of time ACS startup scripts are given to execute. For example, setting this value to “3” would effectively triple the amount of time the acsStart script has to finish before exiting abnormally.

ACS_TMP ($ACSDATA/tmp/, when not set)

The path where all ACS temporary files are written.
This environment variable “cooperates” with the following other variables to allow specifying in a flexible way where ACS temporary files are written:

- ACS_LOG_FILE
- ACS_BACI_RECOVERY_FILE
- ACS_RECOVERY_FILE

The following policy is used to determine where an ACS temporary file is written:

- if none of the above is defined, all files are put in the $ACSDATA/tmp directory
- if ACS_TMP is defined, then temporary files are put in the ACS_TMP directory

- if ACS_LOG_FILE, ACS_BACI_RECOVERY_FILE or ACI_RECOVERY_FILE_NAME is/are defined than file(s) should are created using the env. variable(s) and ACS_TMP is ignored

The utility function `getTempFileName()` allows applications to generate a temporary filename according to this policy.

ALMASW_RELEASE

The current ACS release. This is ACS-<version>.
This is (together with ALMASW_ROOTDIR) the main variable used to derive all other environment variables; for example, ACSROOT is built as:

```
$ALMASW_ROOTDIR/$ALMASW_RELEASE/ACSSW
```

Users can overwrite the default value to switch between different versions of ACS.

ALMASW_ROOTDIR

The root directory for all ALMA ACS software components and all releases.
This is (together with ALMASW_RELEASE) the main variable used to derive all other environment variables.
By default it points to /alma.

INTROOT

The integration area where user software is installed.
Typically each user has their own integration area.
See the ACS Installation Manual1 for details

MANAGER_REFERENCE

CORBA Reference for the MACI Manager (see section Error: Reference source not found for details)

NAMESERVICE_REFERENCE

CORBA Reference for the Naming Service (see section Error: Reference source not found for details)

5.2 Other environment variables set in .bash_profile.acs and used by ACS:

CLASSPATH

form: <JARFILE01>:<JARFILE02>:<JARFILE03>

used by: Java runtime and compile time

Tells Java where to look for Java class files

JAVA_HOME

form: <DIRECTORY>

used by: Java tools

Tells Java/Java tools where to find the Java installation currently being used.

ACE_ROOT

form: <DIRECTORY:$ACE_ROOT_DIR/linux>

used by: ACS environment initialization

This tells where ACE is installed (used by ACE)

ACE_ROOT_DIR

form: <DIRECTORY>

used by: ACS environment initialization

ACS environment variable which tells where the various architecture specific versions of the ACE wrappers can be found

ALMASW_RELEASE

form: <DIRECTORY>

used by: ACS environment initialization

Tells what is the version of the current ACS system

ALMASW_ROOTDIR:

form: <DIRECTORY:/alma>

used by: ACS environment initialization

Tells where the root of all ALMA software is found.

ALMASW_INSTDIR

form: <DIRECTORY:$ALMASW_ROOTDIR/$ALMASW_RELEASE>

used by: ACS environment initialization

Tells where ALMA subsystems are installed.

ANT_HOME

form: <DIRECTORY>

used by: ACS make system

Tells where the ANT make system can be found

OMNI_ROOT

form: <DIRECTORY>

used by: ACS make system

Tells where OmniOrb is installed.

OMNIORB_CONFIG

form: <DIRECTORY:$OMNI_ROOT/config>

used by: OmniOrb

Tells where the configuration files for OmniOrb are found.

IDL_PATH

form: -I<DIRECTORY> -I<DIRECTORY> -I<DIRECTORY>

used by: ACS make system

Where to find IDL files.

JACORB_HOME

form: <DIRECTORY>

used by: ACS make system

Where JacOrb is installed.

PYTHON_ROOT

form: <DIRECTORY>

used by: ACS make system

Where Python is installed

PYTHONPATH

form: <DIRECTORY01>:<DIRECTORY02>:<DIRECTORY03>

used by: Python tools

Where Python source files can be found

GNU_ROOT

form: <DIRECTORY>

used by: <<UNKNOWN>>

Where ACS shipped GNU tools are found.

TCLTK_ROOT

form: <DIRECTORY>

used by: ACS make system

Where ACS shipped TCL/Tk

LD_LIBRARY_PATH

form: <DIRECTORY01>:<DIRECTORY02>:<DIRECTORY03>

used by: dynamic loader, ACS containers

Specifies where share libraries can be found.

PATH

form: <DIRECTORY01>:<DIRECTORY02>:<DIRECTORY03>

used by: Unix shells, ACS make system

Specifies where binaries can be found for shell & makefiles

5.3 Environment variables deprecated or not used any more:

CMM_HOST

form: <IPADDRESS:te13.hq.eso.org>

used by: <<UNKNOWN>>

For ESO CMM Configuration Management System. Not used by ALMA

VLTDATA

form: <DIRECTORY:$ACSDATA>

used by: <<UNKNOWN>>

For VLT development environment. Use ACSDATA instead

VLTROOT

form: <DIRECTORY:$ACSROOT>

used by: <<UNKNOWN>>

For VLT development environment. Use ACSROOT instead

RTAPENV

form: <<UNKNOWN>>

used by: <<UNKNOWN>>

For VLT Configuration database. Not used any more

6 TCP Ports Allocation for ACS

All TAO CORBA Services can locate themselves using TAO Multicast mechanism. This is a very useful feature, but it is not a CORBA standard and it is not usable with multiple services, e.g. more than one Naming Service.

ACS uses a port allocation scheme and API functions are available in C++, Java, Python and Bash shell to retrieve/calculate the port of each ACS service.

See also: http://almasw.hq.eso.org/almasw/bin/view/ACS/AcsPortsAllocation for more details and for a discussions.

In order to allow multiple users to work in completely separate sandboxes on the same machine, we have introduced the concept of “ACS Instance”

The environment variable ACS_INSTANCE can assume the values from 0 (default) to 9.

Based on this environment variable (and/or the –b <ACS_instance #> option for most ACS commands) ports are calculated according to the following formula:

<service port> = 3000 + 100 * ACS_INSTANCE + <offset>

ACS_INSTANCE=0 is treated in a special way for what concerns ports used by Containers: The whole range from 4000 to 4999 is allocated to this purpose. This is done to allow the ACS_INSTANCE=0, to be used in operations and for larger test setups, to handle many more Containers and services than basic developer’s configurations.

Service	offset
Standard Manager	0
CORBA Services	*Range 1 to 10*
Naming Service	1
Notify Service	*2*
Logging Service	3

Interface repository	4
(Logging) notification service	5
(BACI monitor archiving notification service	6
ACS Extended Services	*Range 11 to 20*
ACS Log Service	11
Configuration Database	12
ACS Containers	*Range 4000 to 4999 for ACS_INSTANCE==0, 50-98inside range for other values of ACS_INSTANCE (even numbers only)*

If the Container port passed to a Container startup command is smaller or equal to 24 it is considered an offset in the range of the instance. If it is greater than 24, the TCP port is used as provided.