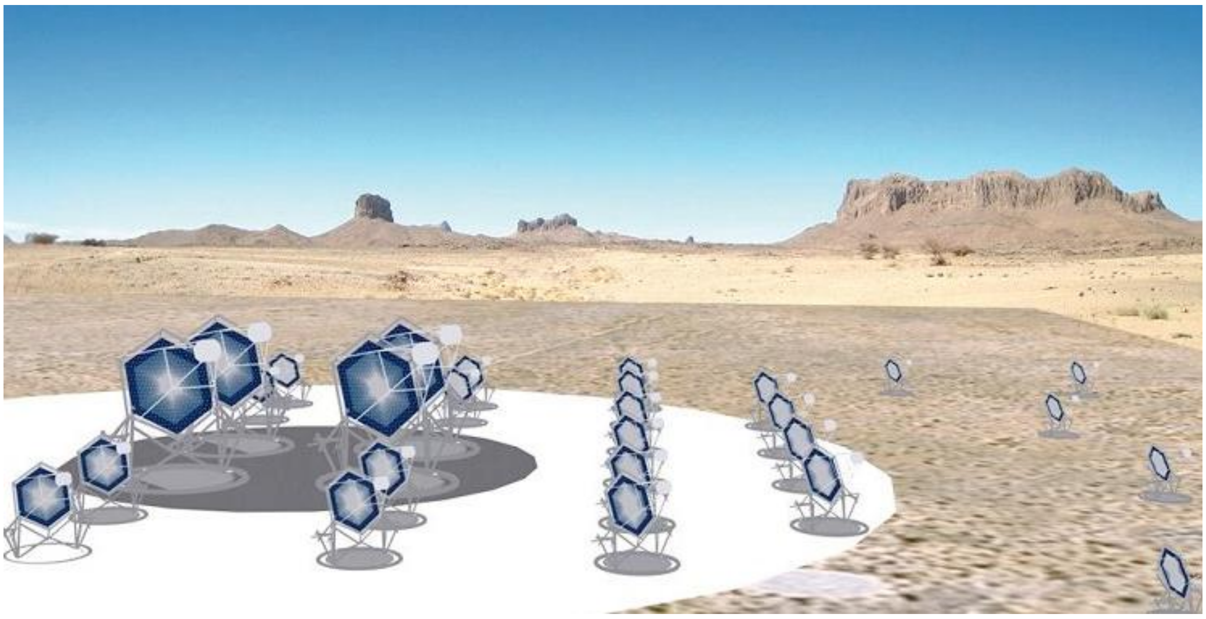
ACS
The first approach



Internal Report IASF Bologna n 614/2013

Prepared by:	Name:	V. Conforti M. Trifoglio A. Bulgarelli F. Gianotti	Signature:	Date:	22/01/2013
Reviewed by:	Name:		Signature:	Date:	
Approved by:	Name:		Signature:	Date:	

TABLE OF CONTENTS

DISTRIBUTION LIST

ASTRI mailing list	astri@brera.inaf.it

DOCUMENT HISTORY

Version	Date	Modification
1.0	22/01/2013	first version
2.0	24/03/2013	First versione revisioned
Draft 3.0	**22/04/2013**	Baci properties and alarms

LIST OF ACRONYMS

ACS	Alma Common Software
IDL	Interface Definition Language
XSD	XML Schema Definition
XML	eXtensible Markup Language
IASFBO	Istituto di Astrofisica Spaziale e Fisica Cosmica - Bologna
MASS	
BACI	BAsic Control Interface
CORBA	Common Object Request Broker Architecture

APPLICABLE DOCUMENTS

REFERENCE DOCUMENTS

[RD1] "Progress report for the activities of the WP3130 A software system for the ASTRI telescope" - Internal Report ASTRI - Stefano Covino - 26/06/2011

[RD2] http://www.eso.org/~almamgr/AlmaAcs/index.html

[RD3] http://www.omg.org/gettingstarted/omg_idl.htm

[RD4] "Common Object Request Broker Architecture (CORBA) Specification, Versione 3.3" - OMG - Novembre 2012

[RD5] http://www.omg.org/spec/CORBA/3.3/

[RD6] http://www.w3schools.com/schema/default.asp

[RD7] "Logging and Archiving" Architecture Document - Klemen Žagar, Radostina Georgieva - 30/07/2007-

[RD8] "ACS Alarm System - Software Architecture and How-to manual", Alessandro Caproni, Bogdan Jeram, 2010-01-13

[RD9] "Mapping CORBA Data types:" http://docs.oracle.com/cd/E13203_01/tuxedo/tux80/cref/member.htm

1. INTRODUCTION

This tutorial is intended for the developers involved in the implementation tasks of the MASS. As presented in [RD1] the needs of a framework which allows the development of the ASTRI software using the distributed programming paradigm, could by satisfied by the ACS. Thus it is very important for all developer teams involved in the project to learn how to use this very powerful system.

2. SCOPE

The main goal of this document is to share our first experience at IASFBO of software development using the ACS framework. In the ASTRI project there are many programmers, each one with his/her own professional background and know how. The use of a framework could require developers to quickly learn an unfamiliar new programming paradigm. We hope that sharing our experience will help other developers to start their work with a tutorial which drives the developer to implement a first ACS component as exercise. This tutorial is intended for C++ developers but its discussion of ACS basics should be useful for Java developers as well.

The following chapters explain all the steps of software implementation starting from the very simple requirements up to and including the testing of the program.

In chapter 3 we discuss the principles of ACS. Chapter 4 and 5 summarizes the requirements and tutorial goals. The setup of ACS and the development environment are described in chapters 6 and 7. Chapter 8 shows how to obtain the skeleton of the application. Chapter 9 defines the interface of this tutorial. Chapter 10 describes how to configure the application. In chapter 11 we show the implementation files. The compile, test and execution steps are discussed in chapters 12, 13, 14 respectively.

Furthermore in this tutorial we discuss some common errors that the developer may encounter. These notes are highlighted with the character: ! .

3. WHAT IS ACS

The ALMA Common Software (ACS) provides a software infrastructure common to all developers of software for the ALMA (Atacama Large Millimeter Array) Project, and consists of a documented collection of common patterns and components, which implement those patterns. ACS is based on a distributed component model which is implemented as CORBA objects in any of the supported programming languages (C++, Java, Python). The teams responsible for the control system's development use ACS Components as the basis for controlling high level entities and for the implementation of devices [RD2].

4. Tutorial Requirements

In this chapter is described the application which the developer must implement following this tutorial.

4.1 The story:

The user wonts an application that is aimed at the simulation of a generic device management. The device has two interface:

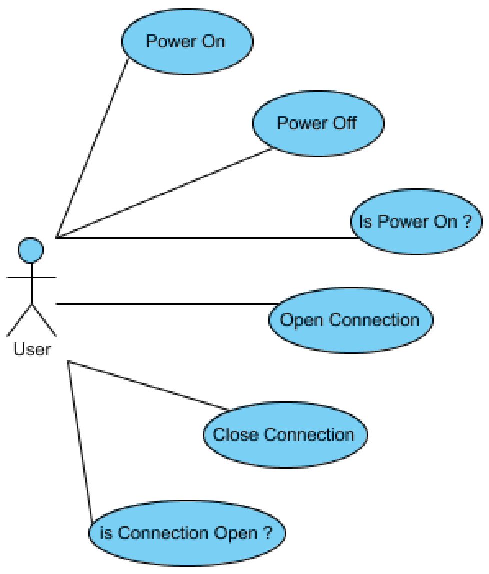
1. The Power;
2. The number of connections;

The power has 3 level, then the admitted values are:

- 0: OFF;
- 1: normal;
- 2: high;

The max number of connections that the device can mangament are 5. When the device must keep more than 5 connection is not ensured the correctness of transmissions;

The followings use cases diagram and tables describe the system requirements:



Use Case: Power On

Name	***Value***
Preconditions	ACS and Container is started up.
Post-conditions	The Power value is incremented.
Flow of Events	
1. Increment the power value;	
2. Notify the new power value;	
3. Read the connections value;	
4. if there are just open connections	
4.1. Notify the system anomaly	
4. end if	

Use Case: Power Off

Name	***Value***
Preconditions	ACS and Container is started up.
Post-conditions	The system is Power Off and All connections is closed.
Flow of Events	
1. Set to 0 the Power value;	
2. Set to 0 The connections value;	

Use Case: Is Power On ?

Name	***Value***
Preconditions	ACS and Container is started up.
Post-conditions	Return true if the system is power on, false otherwise
Flow of Events	
1. Read the Power value	
2. if the Power value is greater than 0	

Name	*Value*
2.1. return TRUE	
3. else	
3.1. return FALSE	
3. end if	

Use Case: Open Connection

Name	*Value*
Preconditions	ACS and Container is started up.
Post-conditions	The connection value is incremented
Flow of Events	
1. Read the current connection value;	
2. Increment the current connection value;	
3. Update the connection value.	

Use Case: Close Connection

Name	*Value*
Preconditions	ACS and Container is started up.
Post-conditions	the connection value is decremented.
Flow of Events	
1. Read the current connection value;	
2. Decrement the current connection value;	
3. Update the connection value.	

Use Case: is Connection Open ?

Name	*Value*
Preconditions	ACS and Container is started up.

Name	*Value*
Post-conditions	Return true if the are open connections, false otherwise
Flow of Events	
1. read the connections value	
2. if The connections value is greater than 0	
2.1. Return true	
3. else	
3.1. Return false	
3. end if	

5. ACS Learning goals

At the end of this tutorial the developer will be acquire the following know how:

1. The setup of the ACS environment;
2. The definition of IDL Interfaces of a component;
3. The development of a Characteristic Component;
4. The management of BACI properties to monitor a device;
5. The ACS alarm definitions.

6. ACS Setup

This tutorial does not explain how to install ACS but aims at helping developers write their first ACS program very quickly.

A virtual machine with ACS and other useful tools installed is available for the (free) Oracle Virtual Box, 4.1.22 or later.

This machine is configured with 3 user accounts. For this tutorial only the user *ctadev* is needed and its password is *123456*.

The programs installed on the virtual machine are the following:

- Eclipse 4.2.1 with plugins that support C++, Python, Git and Svn;
- Java 1.6 update 37;
- Git 1.7;
- SVN 1.6.x;

The virtual machine can be downloaded from this web address:
`ftp://astrisw_ftp@ciws.iasfbo.inaf.it/VMS/VboxSL6.3ACS10.2.tar.gz`

Other instructions and updates are published on the redmine web site: http://redmine.iasfbo.inaf.it/projects/astri-acs.

7. Environment setup

After the first setup of ACS, the environment for the specific project must be configured. This operation will be done one time for each project.

For compilation and testing, the developer must create a directory called INTROOT (for “integration root”). The directory is populated with the results of the build (compile, link & install) process; the developer never writes directly in this directory. The INTROOT directory will be used by all developers involved in the project as the space where all components and clients be integrated.

To create the introot, run the following command:

```
getTemplateForDirectory INTROOT <path to directory>
```

Where `<path to directory>` is the full path (directory and file name) to be assigned to the INTROOT directory.

Then you can proceed with setting the bash configuration in order to set the environment variables that ACS uses to point to the INTROOT directory just created.

Open the file .bashrc in the home directory with a text editor and append the following text:

```
export INTROOT=$HOME/introot # or the path you have chosen
source $HOME/.acs/.bash_profile.acs
```

To apply the new setup to the current shell, run the command:

```
source .bashrc
```

Now the environment is set up both for the current shell and for the new shells that you will run in the machine.

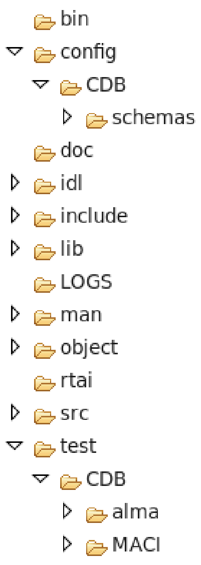
8. Create the Directory Structure

ACS provides a function to create the directory structure for a module.

Once you have decided where to develop and work with the code for your module, run the command:

```
getTemplateForDirectory MODROOT_WS <path to directory>
```

In this example we have named the module "ACSdps". An example of the created directory structure is shown in the following figure:



```
  bin
▽ config
  ▽ CDB
    ▷ schemas
  doc
▷ idl
▷ include
▷ lib
  LOGS
▷ man
▷ object
  rtai
▷ src
▽ test
  ▽ CDB
    ▷ alma
    ▷ MACI
```

Figure 1 – Directory tree of the project

These directories will host all source and compiled code. The program will be installed in the INTROOT directory after a successful build.

The purpose of each of these subdirectories will become clearer as the tutorial proceeds. The INTROOT directory will hold the build artifacts of all components and it will ensure that ACS will be able to find and manage them at runtime. It is worth noting that the directories named “idl” usually contain interface definitions that enable each component to call a method of another component just knowing its interface.

We will take advantage of this feature in the tutorial when we use the Object Explorer, a generic tool for running any ACS component, to test the component that we develop.

9. Define the IDL interface

The IDL interface defines the functionality that a component offers to its clients, which may be (but are not necessarily) themselves components. The interface is the only mode for a component to tell clients the methods that it exposes. Likewise a component knowing the interface of the other component can use the other component's methods without knowing how (or even in what programming language) the component is implemented.

The IDL interface must be written in a file with an .idl or .midl extension and must be put into the *idl* folder.

The interface for the component we are developing in this tutorial is:

```
#ifndef _DPS2_IDL_
#define _DPS2_IDL_

#pragma prefix "alma"

#include <baci.idl>
#include <acscommon.idl>

module DPSModule {

        interface DPSInterface:ACS::CharacteristicComponent {

                /**
                 * Power On the system
                 *
                 * @Action
                 */
                void powerOn();

                /**
                 * Power off the system
                 *
                 * @Action
                 */
                void powerOff();

                /**
                 * check the status of the system power
                 *
                 * @return 1 if the status is on, 0 otherwise
                 *
                 * @Action
                 */
                //ACS::ROdouble isPowerOn();
                boolean isPowerOn();


                /**
                 * open a connection
                 *
                 * @Action
                 */
                void openConnection();

                /**
```

```
                         * close a connection
                         *
                         * @Action
                         */
                        void closeConnection();

                        /*
                         * check the status of the connection
                         *
                         *
                         * @return 1 if there are  connections opened , 0 otherwise
                         * @Action
                         */
                        //ACS::ROdouble isConnectionOpen();
                        boolean isConnectionOpen();



                        // the following properties are use to know the status of the system
                        readonly attribute ACS::ROdouble powerStatus; // @ Property
                        readonly attribute ACS::ROdouble connectionStatus; // @ Property

                };
        };
#endif
```

In this interface we import the interface definitions for two other components: **baci** and **acscommon**. These components are part of the ACS framework and provide many features. It is possible to view the services offered by these ACS components by opening the interface definition files contained in the ACS core (for the ACS version in this tutorial the absolute path is : /alma/ACS-10.2/ACSSW/idl/{baci.idl and acscommon.id}).

This interface defines the interfaces and methods that will be made available to clients. For this tutorial there is only one module with only one interface exposing the *displayMessage* method.

The ACS framework allows the interaction between the component and the device providing the BACI (BAsic Control Interface) properties. When we wont use the BACI properties then the our interface must be inherit from the "CharacteristicComponent" ACS class. In this tutorial as defined in the above IDL file we use two BACI properties:

1. powerStatus: it keeps track of power level;
2. connectionStatus: it keeps track of number connections opened;

It is important to notice that the IDL syntax must always be respected. To learn more on the IDL concepts see [RD3] and [RD4].

When many teams are working for a common goal, the definition of the interfaces is crucial and needs several iterative discussions between all team groups in order to agree on a common solution.

10. Configuration of the Application

The application's BACI properties areconfigured via an XML schema and a corresponding XML instance document. The schema typically defines defaults for the values, error/alarm limits and monitoring rates for the properties, while the instance document can override them. These files will also be useful for testing.

The schema file must have an *.xsd* extension and must be created in the following path:

..ACSdps/config/CDB/schemas/dpsSchema.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema                    targetNamespace="urn:schemas-cosylab-
com:dpsSchema:1.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns="urn:schemas-cosylab-com:dpsSchema:1.0"
         xmlns:cdb="urn:schemas-cosylab-com:CDB:1.0"
         xmlns:baci="urn:schemas-cosylab-com:BACI:1.0"

         elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:import            namespace="urn:schemas-cosylab-com:CDB:1.0"
schemaLocation="CDB.xsd"/>
<xs:import           namespace="urn:schemas-cosylab-com:BACI:1.0"
schemaLocation="BACI.xsd"/>
<xs:complexType name="dpsSchema">
     <xs:complexContent>
          <xs:extension base="baci:CharacteristicComponent">
               <xs:sequence>
                    <xs:element                name="getStatus"
type="baci:ROdouble" />
                    <xs:element                name="setStatus"
type="baci:ROdouble" />
               </xs:sequence>
          </xs:extension>
```

```
    </xs:complexContent>
</xs:complexType>
<xs:element name="dpsSchema" type="dpsSchema"/>
</xs:schema>
```

The contents of this file are the namespaces, the declaration of a new complex type dpschema which contains complex content which extends a CharacteristicComponent. The sequence tag encloses the two BACI properties defined in the IDL file.

The writing of this schema must be accurate because it exposes the rules which must be respected by the other xml files used for testing software.

Further details about the XSD standard can be found in [RD6].

! A common error is to misspell the names of methods, variables, types (anything that needs a name) that refer to the same object in different files. For example in this tutorial we used a property named getStatus. It is crucial to remember that the first character is lowercase, the *S* of Status is uppercase and there are no underscore characters. The use of a consistent convention for clear code object naming helps a lot to avoid errors which could be very hard to find at compile time.

11. Program implementation

In this tutorial the program is implemented using the C++ language, but also Java and Python are accepted by ACS.

The C++ implementation requires the header file and the implementation file: *ACSdpsImpl.h* and *ACSdpsImpl.cpp*.

The *ACSdpsImpl.h* must be put in the *include* folder:

```
#ifndef ACSdps2Impl_h
#define ACSdps2Impl_h
#endif

#ifndef __cplusplus
#error This is a C++ include file and cannot be used from plain C
#endif


// This Stub is automatically created by ACS. It is needed.
#include "ACSdps2S.h"

// This component is provided by ACS
#include <baciCharacteristicComponentImpl.h>
#include <acsexmplExport.h>


///Includes for each BACI property used in this example
#include <baciROdouble.h>

///Include the smart pointer for properties
#include <baciSmartPropertyPointer.h>

//DevIO allow the wreating and reading of the baci properties.
#include <baciDevIOMem.h>



/*
 *
 * \brief The class DPS is aimed at the interaction with the device
 *
 */
class   DPSinterface: public virtual baci::CharacteristicComponentImpl, public virtual
POA_DPSModule::DPSInterface {

public:
        // Constructor
        DPSinterface(const       ACE_CString&       name,       maci::ContainerServices*
containerServices);


        // Destructor
        virtual ~DPSinterface();

        /*------------------- [CORBA interface ] ------------------------- */

        /**
         * Power On the system
         *
         * @Action change the value of the powerStatus baci properties
         */
        virtual void powerOn();

        /**
         * Power off the system
         *
         * @Action
         */
        virtual void  powerOff();

        /**
         * check the status of the dps power
         *
         * @return 1 if the status is on, 0 otherwise
         *
         * @Action
         */
        virtual CORBA::Boolean isPowerOn();

        /**
         * open a connection
         *
         * @Action
         */
        virtual void openConnection();

        /**
         * close a connection
         *
         * @Action
         */
        virtual void closeConnection();

        /*
         * check the status of the connection
         *
         *
         * @return 1 if the connections is open, 0 otherwise
         * @Action
         */
        virtual CORBA::Boolean isConnectionOpen();


        //definition of baci properties pointer
        virtual ACS::ROdouble_ptr powerStatus();
        virtual ACS::ROdouble_ptr connectionStatus();

        baci::SmartPropertyPointer<baci::ROdouble> m_powerStatus_sp;
        baci::SmartPropertyPointer<baci::ROdouble> m_connectionStatus_sp;

};
```

The name of the class must be the same as the interface defined in the IDL file. This class must inherit from *CharacteristicComponentImpl*, as defined in the idl file. Since it is the implementation file, here must be inserted always the implementations of the parents components. There is a naming convention that requires that the implementation of a component has the suffix "Impl". It is always also required that the implementation class inherits the idl interface which is built as:

```
POA_module::interface
```

Where *module* and *interface* are defined in the idl file.

In our case:

```
POA_ACSdps::DPSInterface
```

In the above code is shown how must be declared the BACI properties (*getStatus* and *setStatus*). All the CORBA details are hidden by the ACS that allows the developer to declares easily the properties and them pointers.

In case of methods which have I/O parameters is needed to pay close attention at the type syntax parameters. ACS allows the interaction with the etherogeneous component using the CORBA middleweare. The developer does not interact directly with CORBA but the interaction between the IDL and the our component is precisely CORBA. Thus the developer must use the rigth syntax for the IDL type, CORBA type and his language programming type (C++ for this tutorial) [RD9].

For example if in IDL it is defined a boolean value as return of a function, the developer shall use the type "`boolean`" in the IDL interface and the type "`CORBA::Boolean`" in the headers function of the C++ source code.

The file *ACSdpsImpl.cpp* must be put in the *src* folder.

For what the implementation is concerned:

```
ifndef __cplusplus
#error This is a C++ include file and cannot be used from plain
C
#endif

#include <ACSdps2Impl.h>
#include <ACSdps2S.h>

#include <maciContainerServices.h>
#include <logging.h>
#include <acsutil.h>
#include <maciACSComponentDefines.h>

using namespace baci;
using namespace std;

MACI_DLL_SUPPORT_FUNCTIONS (DPSinterface)
```

```
// Implementation of the constructor
DPSinterface::DPSinterface(const ACE_CString& name,
maci::ContainerServices*
containerServices):CharacteristicComponentImpl(name,
containerServices),m_powerStatus_sp(new
ROdouble(name+":powerStatus", getComponent()),this) ,
m_connectionStatus_sp(new ROdouble(name+":connectionStatus",
getComponent()),this) {

      // set the value of the powerStatus and connectionStatus to
0
      ACS::Time timestamp;
      double value = 0.0;
      m_powerStatus_sp->getDevIO()->write( value, timestamp);
      m_connectionStatus_sp->getDevIO()->write(            value,
timestamp);


      // creation of log of the constructor
      ACS_TRACE("::DPS::DPS ... constructor ... done.");

}


// Implementation of distructor
DPSinterface::~DPSinterface(){

      // creation of log of the denstructor
      ACS_TRACE("::DPS::DPS ... distructor ... done.");
}




/*
 *
 * POWER ON
 *
 * \brief increment the value of the powerStatus. Log a warning
if  the first is power          on and there is just an open
connection.
 *
 *
 */
      void DPSinterface::powerOn(){

            ACS_SHORT_LOG((LM_INFO, "Method called: power On "));
            ACS::Time timestamp;
            // read powerStatus and connectionStatus
            double powerStatus = m_powerStatus_sp->getDevIO()-
>read(timestamp);
            double connectionStatus = m_connectionStatus_sp-
>getDevIO()->read(timestamp);

            // check closedConnection
            if ( (connectionStatus != 0) && (powerStatus == 0)
){
                  ACS_SHORT_LOG((LM_INFO, "WARNING: The current
Connection Status is: %f", connectionStatus));
            }


            // increment the value of the powerStatus
            double value = powerStatus + 1.0;
            m_powerStatus_sp->getDevIO()->write( value,
timestamp);
            // read the value just set and log it
            powerStatus = m_powerStatus_sp->getDevIO()-
>read(timestamp);
            ACS_SHORT_LOG((LM_INFO, "The current Power Status is:
%f", powerStatus));
      }



      /*
       *
       * POWER OFF
       *
       * \brief Set the connectionStatus and powerStatus to 0
       *
       *
       */
      void DPSinterface::powerOff(){

            ACS_SHORT_LOG((LM_INFO,  "Method  called:  power  Off
"));

            ACS::Time timestamp;

            // shutdown all system
            double value = 0.0;
            m_powerStatus_sp->getDevIO()->write(          value,
timestamp);
```

```
            m_connectionStatus_sp->getDevIO()->write(        value,
timestamp);


            // read current status of the system
            // read powerStatus and connectionStatus
            double  powerStatus  =  m_powerStatus_sp->getDevIO()-
>read(timestamp);
            double   connectionStatus   =   m_connectionStatus_sp-
>getDevIO()->read(timestamp);


            ACS_SHORT_LOG((LM_INFO,  "System  shutdown  ->  Power
Status: %f ", powerStatus));
            ACS_SHORT_LOG((LM_INFO,    "System    shutdown    ->
Connection Status: %f " , connectionStatus));
              /*

      }



      /*
       *
       *    IS POWER ON
       *
       *    \return true if the camera server is power on, false
otherwise
       */
      CORBA::Boolean DPSinterface::isPowerOn(){

            ACS_SHORT_LOG((LM_INFO, "Method called: is power on ?
"));

            ACS::Time timestamp;
            double  powerStatus  =  m_powerStatus_sp->getDevIO()-
>read(timestamp);

            bool status = false;
            if (powerStatus > 0 ){
                  status = true;
            }

            return status;
      }
```

```
        /*
         *
         *    OPEN CONNECTION
         *
         *    \brief increment the number of connections
         */
        void DPSinterface::openConnection(){

                ACS_SHORT_LOG((LM_INFO,    "Method    called:    open
connection"));

                // read current connection opened
                ACS::Time timestamp;
                double connectionStatus = m_connectionStatus_sp-
>getDevIO()->read(timestamp);

                // increment the number of connections
                connectionStatus++;
                m_connectionStatus_sp->getDevIO()-
>write(connectionStatus, timestamp);

                ACS_SHORT_LOG((LM_INFO,   "The   current   Connection
opended are: %f", connectionStatus));


        }

        /*
         *
         *    CLOSE CONNECTION
         *
         *    \brief Decrement the number of connections
         */
        void DPSinterface::closeConnection(){

                ACS_SHORT_LOG((LM_INFO, "Method called: close
connection"));

                // read current connection opened
                ACS::Time timestamp;
                double connectionStatus = m_connectionStatus_sp-
>getDevIO()->read(timestamp);

                // increment the number of connections
                connectionStatus--;
                m_connectionStatus_sp->getDevIO()-
>write(connectionStatus, timestamp);
```

```
            ACS_SHORT_LOG((LM_INFO, "The current Connection
opended are: %f", connectionStatus));

      }


      /*
       *
       *     IS CONNECTION OPENED
       *
       *    \return true if there are connections opened. False
otherwise
       */
      CORBA::Boolean DPSinterface::isConnectionOpen(){

            ACS_SHORT_LOG((LM_INFO, "Method called: is connection
open ? "));

            ACS::Time timestamp;
            double connectionStatus = m_connectionStatus_sp-
>getDevIO()->read(timestamp);

            bool status = false;
            if (connectionStatus > 0 ){
                  status = true;
            }

            return status;
      }


      /* --------------------- [ CORBA interface ] --------------
---------*/


      ACS::ROdouble_ptr DPSinterface::powerStatus(){

            if (m_powerStatus_sp == 0){
                        return ACS::ROdouble::_nil();
            }
            ACS::ROdouble_var prop =
ACS::ROdouble::_narrow(m_powerStatus_sp->getCORBAReference());
            return prop._retn();
      }


      ACS::ROdouble_ptr DPSinterface::connectionStatus()
      {
          if (m_connectionStatus_sp == 0)
           {
           return ACS::ROdouble::_nil();
           }

          ACS::ROdouble_var prop =
ACS::ROdouble::_narrow(m_connectionStatus_sp-
>getCORBAReference());
          return prop._retn();
      }
```

The methods implemented satisfy the requirements presented in the chapter 4:

1. power On: it increments the `powerStatus` value and check that that are any connections opened if it is the first call of powerOn method;
2. power Off: it set the `powerStatus` and the `connectionStatus` values to 0;
3. isPowerOn: it returns true if the `powerStatus` value is greather than 0, false otherwise;
4. openConnection: it increments the `connectionStatus` value;
5. closeConnection: it decrements the `connectionStatus` value;
6. isConnectionOpen: it returns true if the `connectionStatus` value is greather than 0, false otherwise;

There are also the following methods:

- Constructor: it set the `powerStatus` and `connectionStatus` to 0;
- Distructor: it make a log of method called;
- CORBA interfaces methods: they implements the methods to manage the baci pointers;

In the above implementation it is note that it is used the standard DEVIO of the ACS for the reading and writing of the baci properties value. In case of the hardware device the developer must implement the DevIO in order to interface the component with the real hardware device.

The following instruction for example allow to execute a writing operation:

```
m_powerStatus_sp->getDevIO()->write( value, timestamp);
```

The DevIO is used as method of the baci pointer and it has only two method:

1. read;
2. write;

In both cases it is neede to pass the timestamp to keep track when the method is called.

Both the constructor and destructor use an ACS_TRACE macro (which produces a log message at TRACE level) to keep track of the status of the execution. The parameters passed to the constructor will be useful to the ACS manager and the container.

In the any methods it is used the ACS logging macro which generates a log with a simple text message. The syntax is:

```
ACS_SHORT_LOG((LM_INFO, "text of log "));
```

For complete syntax and functional details, see [RD7].

The *CORBA interface* section is where we implement the interface methods (including any BACI properties) defined in the original IDL file.

! Care should be taken in the declaration of the constructor to not forget anything just declared in .h file, to spell all variable names correctly and to apply the correct operations to BACI properties (e.g., by not attempting to write to a read only property).

!! It must be the maxim attention when change a name or a type of a variable because often it involves tha changment of more than one file. Then check always all the files (idl, xsd, xml, h, cpp/j/py) . It may seem like a stupid advice but these are the main causes of error.

12. Compile using the Makefile

Now all is ready for program compiling. When ACS created the directory structure, it also created a standard makefile, located in *src* folder, that can be adapted to our case. In the following we show the modified Makefile. Added lines are preceded by a string like " `#ADDED BY HELLO WORLD TUTORIAL` ":

```
#****************************************************************
****************
#
#
# "@(#) $Id$"
#
# Makefile of ACSdps
#
# who       when      what
# Conforti  10/04/13  dps component
# ctadev    10/04/13  created
#


#
# user definable C-compilation flags
#USER_CFLAGS =

#
# additional include and library search paths
#USER_INC =

#ADDED BY HELLO WORLD TUTORIAL
USER_LIB = -lACE \
         -lTAO \
         -lTAO_DsLogAdmin \
         -lTAO_CosNaming \
         -lTAO_IORTable \
         -lTAO_PortableServer \
         -lTAO_Svc_Utils \
         -lTAO_CosTrading \
         -lTAO_CosNotification \
         -lTAO_DynamicAny \
         -lTAO_IFR_Client \
         -lTAO_CosProperty \
         -lacsutil \
         -lcdb \
         -llogging \
         -lacscomponent \
         -lbaci \
          -lmaci \
          -lacsErrTypeComponent \
          -lmaciClient \
          -lacserr \
          -lm \
          -lloki \
          -lacstime
#
# MODULE CODE DESCRIPTION:
# ------------------------
# As a general rule:  public file are "cleaned" and "installed"
#                     local (_L) are not "installed".

#
# C programs (public and local)
# -----------------------------
EXECUTABLES     =
EXECUTABLES_L   =

#
# <brief description of xxxxx program>
xxxxx_OBJECTS   =
xxxxx_LDFLAGS   =
xxxxx_LIBS      =

#
# special compilation flags for single c sources
#yyyyy_CFLAGS   =

#
# Includes (.h) files (public only)
# ---------------------------------
INCLUDES        =    ACSdps2Impl.h

#
# Libraries (public and local)
# ----------------------------
LIBRARIES       =    ACSdps2Impl
LIBRARIES_L     =

ACSdps2Impl_OBJECTS = ACSdps2Impl
ACSdps2Impl_LIBS = ACSdps2Stubs maci logging acslogStubs

#
# <brief description of lllll library>
lllll_OBJECTS   =

#
```

```
# Scripts (public and local)
# ----------------------------
SCRIPTS         =
SCRIPTS_L       =

#
# TCL scripts (public and local)
# ------------------------------
TCL_SCRIPTS     =
TCL_SCRIPTS_L   =

#
# Python stuff (public and local)
# ----------------------------
PY_SCRIPTS         =
PY_SCRIPTS_L       =

PY_MODULES         =
PY_MODULES_L       =

PY_PACKAGES        =
PY_PACKAGES_L      =
pppppp_MODULES     =

#
# <brief description of tttttt tcl-script>
tttttt_OBJECTS  =
tttttt_TCLSH    =
tttttt_LIBS     =

#
# TCL libraries (public and local)
# ------------------------------
TCL_LIBRARIES   =
TCL_LIBRARIES_L =

#
# <brief description of tttlll library>
tttlll_OBJECTS  =

#
# Configuration Database Files
# ----------------------------
CDB_SCHEMAS = dpsSchema

#
# IDL Files and flags
#
```

```
IDL_FILES =           ACSdps2
TAO_IDLFLAGS =
USER_IDL =
#
# Jarfiles and their directories
#
JARFILES=
jjj_DIRS=
jjj_EXTRAS=
# For expressing dependencies between jarfiles (parallel builds)
jjj_JLIBS=
#
# java sources in Jarfile on/off
DEBUG=
#
# ACS XmlIdl generation on/off
#
XML_IDL=
#
# Java Component Helper Classes generation on/off
#
COMPONENT_HELPERS=
#
# Java Entity Classes generation on/off
#
XSDBIND=
#
# Schema Config files for the above
#
XSDBIND_INCLUDE=
# man pages to be done
# --------------------
MANSECTIONS =
MAN1 =
MAN3 =
MAN5 =
MAN7 =
MAN8 =

#
# local man pages
# ---------------
MANl =

#
# ASCII file to be converted into Framemaker-MIF
# --------------------
ASCII_TO_MIF =
#
# other files to be installed
#----------------------------
INSTALL_FILES =

#
# list of all possible C-sources (used to create automatic
dependencies)
# ------------------------------
CSOURCENAMES = \
      $(foreach exe, $(EXECUTABLES) $(EXECUTABLES_L),
$($(exe)_OBJECTS)) \
      $(foreach rtos, $(RTAI_MODULES) , $($(rtos)_OBJECTS)) \
      $(foreach lib, $(LIBRARIES) $(LIBRARIES_L),
$($(lib)_OBJECTS))

#
#>>>>> END OF standard rules

#
# INCLUDE STANDARDS
# -----------------

MAKEDIRTMP := $(shell searchFile include/acsMakefile)
ifneq ($(MAKEDIRTMP),\#error\#)
   MAKEDIR := $(MAKEDIRTMP)/include
   include $(MAKEDIR)/acsMakefile
endif

#
# TARGETS
# -------
all:  do_all
      @echo " . . . 'all' done"

clean : clean_all
      @echo " . . . clean done"

clean_dist : clean_all clean_dist_all
      @echo " . . . clean_dist done"

man   : do_man
      @echo " . . . man page(s) done"

install : install_all
      @echo " . . . installation done"
```

#___oOo___

It is noted that additions are required for:

- the implementation files
- the idl files for the interfaces
- the ACS core components used in the implementation (logging, MACI, ...)
- the stubs created by ACS

13. Create the code for the testing

This step is required in order to prepare the test that checks both the interfaces and the implementation of the program. It the test is successful it could be needed to copy all the files inside the test/CDB folder under the config/CDB folder in order to execute the configuration of the database (if it is used). The folder involved is **test/CDB** with the following structure:

```
test
        CDB
                Alarms
                        Administrative
                                AlarmSystemConfiguration
                                        AlarmSystemConfiguration.xml
                                Categories
                                        Categories.xml
                                ReductionDefinitions
                                        ReductionDefinition.xml
                        AlarmDefinitions
                                BACIProperty
                                        BACIProperty.xml
                                BACIPropTest#testDoubleVar
                                        BACIPropTest#testDoubleVar.xml
                                DPSInterface
                                        DPSInterface.xml
                                Manager
                                        Manager.xml
                                TestFF
                                        TestFF.xml

                alma
                        TEST_DPS_1
                                TEST_DPS_1.xml
                MACI
                        Components
                                Components.xml
                        Containers
                                dpsContainer
                                        dpsContainer.xml
                        Managers
                                Manager
                                        Manager.xml
```

In the MACI branch there are the definitions of the component, the container, and the manager. The alma branch holds the configuration files for the component instances. The Alarms branch define the configuration of the alarm system (the content of these files is in Annex A and can be copied less any edit in the tutorial exercise workspace). All these files are shown below.

TEST_DPS_1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<dpsSchema xmlns="urn:schemas-cosylab-com:dpsSchema:1.0"
        xmlns:cdb="urn:schemas-cosylab-com:CDB:1.0"
        xmlns:baci="urn:schemas-cosylab-com:BACI:1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

     <powerStatus default_value="1.0" units="int"
description="status of power of the camera server"
alarm_low_on="-1.0" alarm_low_off="0.0" alarm_high_on="2.0"
alarm_high_off="1.0" min_delta_trig="0.1" min_step="0.1"
alarm_timer_trig="1.0" />

     <connectionStatus default_value="1.0" units="int"
description="status of power of the camera server"
alarm_low_on="-1.0" alarm_low_off="0.0" alarm_high_on="6.0"
alarm_high_off="5.0"  min_delta_trig="0.1" min_step="0.1"
alarm_timer_trig="1.0" />
</dpsSchema>
```

In the above configuration file is set the baci properties characteristic. Any default values from the schema may be overridden.

It is note that for the connectionStatus ther is an alarm if the value is less or equal to -1 (alarm_low_on) and this alarm is shut down when the powerStatus value increseas to 0 or greater (alarm_low_off). Similarly the alarm_high_on is 6 and alarm_high_of is 5 because the max number of the opened connections is 5.

Components.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Components xmlns="urn:schemas-cosylab-com:Components:1.0"
xmlns:cdb="urn:schemas-cosylab-com:CDB:1.0"
xmlns:baci="urn:schemas-cosylab-com:BACI:1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
     <_ Name="TEST_DPS_1"
     Code="ACSdps2Impl"
```

```
       Type="IDL:alma/DPSModule/DPSInterface:1.0"
       ImplLang="cpp"
       Container="dpsContainer" />
</Components>
```

In this file is specified the Name of the component instance, *i.e.* TEST_DPS_1 (that is the same defined under *alma* folder), the Code is the name of the compiled DLL file without its extension (ACSdps2Impl), the Type is name of the interface (defined in the IDL file), the Container is the name of the container in which the component will run and ImplLang is the programming language used in the implementation of both the component and the container, in this case, C++.

dpsContainer.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Container    xmlns:cdb="urn:schemas-cosylab-com:CDB:1.0"    xmlns="urn:schemas-cosylab-com:Container:1.0"                          xmlns:baci="urn:schemas-cosylab-com:BACI:1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"    xmlns:log="urn:schemas-cosylab-com:LoggingConfig:1.0" Timeout="20000" UseIFR="1" ImplLang="cpp">
<Autoload>
       <cdb:_ string="baci"/>
</Autoload>
  <LoggingConfig
            centralizedLogger="Log"
            minLogLevel="2"
            dispatchPacketSize="10"
            immediateDispatchLevel="99">
  </LoggingConfig>
</Container>
```

Manager.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Manager xmlns:cdb="urn:schemas-cosylab-com:CDB:1.0"
xmlns="urn:schemas-cosylab-com:Manager:1.0"
xmlns:baci="urn:schemas-cosylab-com:BACI:1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
Timeout="50000" xmlns:log="urn:schemas-cosylab-
com:LoggingConfig:1.0" >
                  <Startup>
                    <cdb:_ string="CLOCK1" />
                    <cdb:_ string="TIMER1" />
                    <cdb:_ string="MOUNT1" />
                  </Startup>
                    <ServiceComponents>
                              <cdb:_ string="Log" />
                              <cdb:_ string="LogFactory" />
```

```
                                <cdb:_
string="NotifyEventChannelFactory" />
                                <cdb:_ string="ArchivingChannel"
/>
                                <cdb:_ string="LoggingChannel"
/>
                                <cdb:_
string="InterfaceRepository" />
                                <cdb:_ string="CDB" />
                                <cdb:_ string="ACSLogSvc" />
                                <cdb:_ string="PDB" />
                                <cdb:_
string="AcsAlarmService"/>
                     </ServiceComponents>

                     <LoggingConfig>
                          <log:_ Name="jacorb@Manager"
minLogLevel="5" minLogLevelLocal="4" />
                     </LoggingConfig>
</Manager>
```

14. Test the program

Now you are ready to test the program.

First the program must be compiled. From the *src* folder execute the following command:

```
make clean all install
```

The *clean* ensures deletion of any result of previously compilations. The *all* compiles the program and the *install* copies all necessary files into the introot directory.

! The compile step can notice will flag syntax errors, including names of undeclared or misspelled variables. In the console it checks the first occurrence of an error (the latter could be caused by the primer).

When the compile success, in order to load the test in ACS, execute the following command under the *test* directory:

```
export ACS_CDB=$PWD
```

(note: if this step is not executed, ACS will load the ACS test example)

Before running the program it is possible to check the correctness of xml test files using the following command:

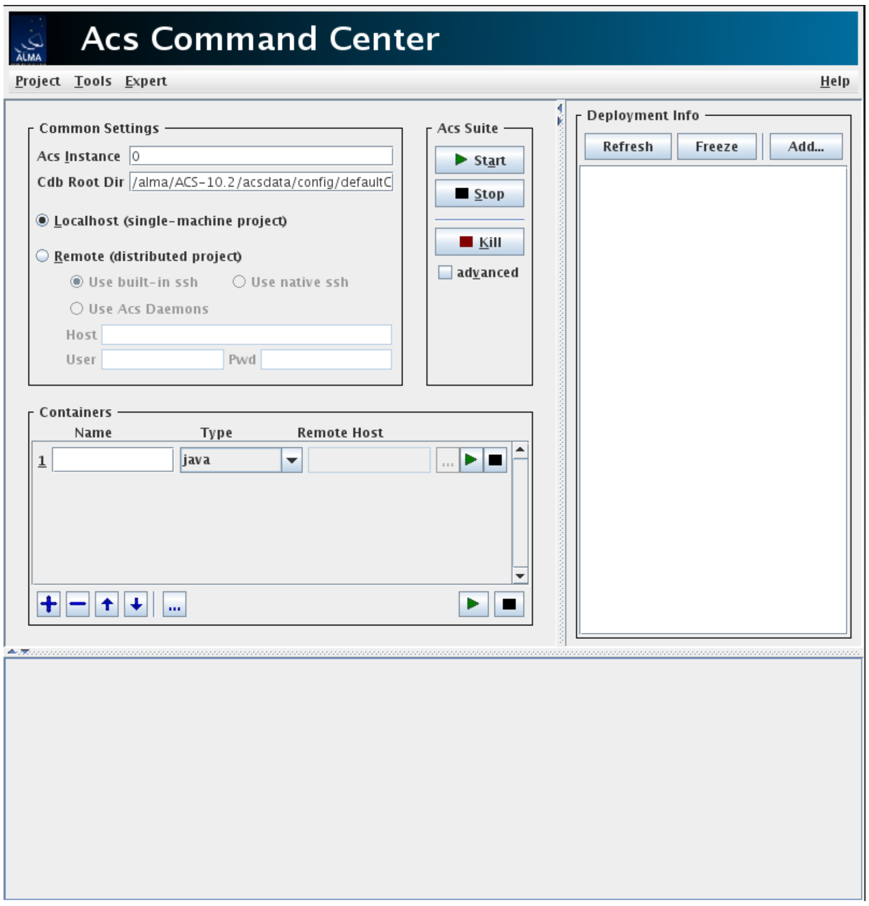
```
cdbChecker
```

This tool is useful for find the error in XML files. Check always the correctness of the words.

If this test succeeds, proceed with the execution of the program. The program can be started using the GUI or the command line. In this tutorial is used the GUI with the following command:

```
acscommandcenter
```

The GUI is shown in the following figure:



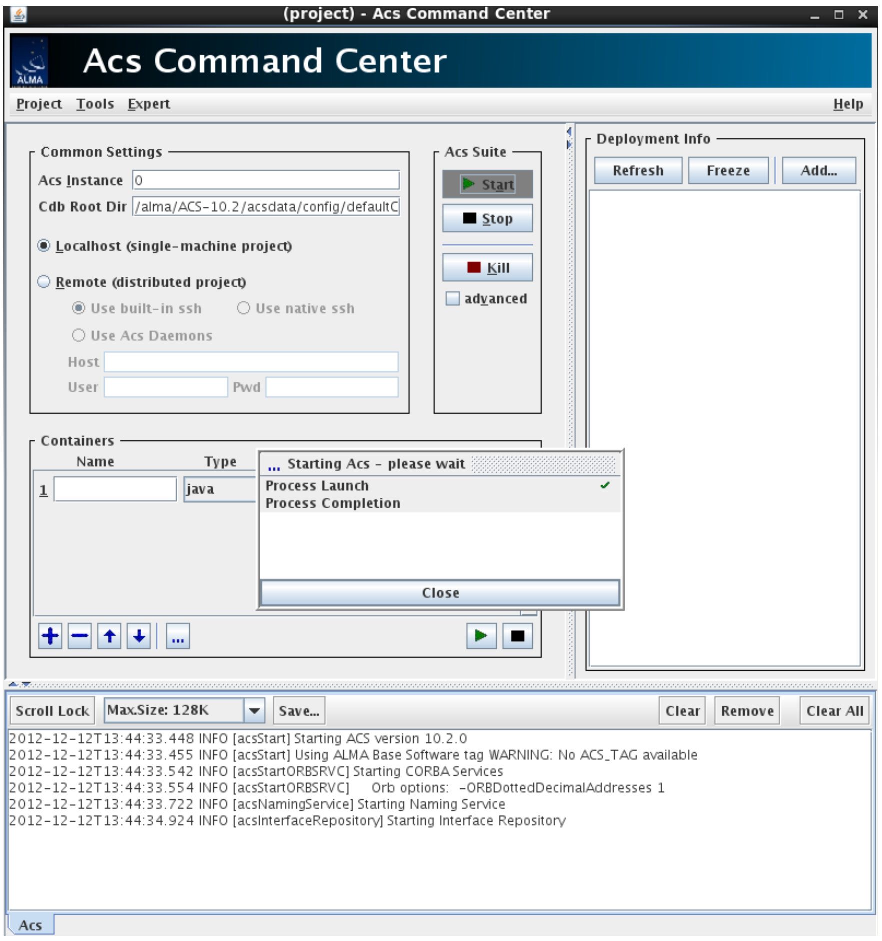
Acs Instance: 0
Cdb Root Dir: /alma/ACS-10.2/acsdata/config/defaultC
(● Localhost (single-machine project))
1 | java

Figure 2 - ACS Main Panel

There are 4 areas:

1. Common settings: it is used for start/stop/kill ACS and its services;
2. Containers: it is used to select, start and stop one or more containers;
3. Deployment info: it is used to show the statuses of containers and components in recognized by the system;
4. Console: it displays every change of status of the ACS (included errors).

When "start" button is selected, a little box appears and it must be waited for the end of the process:



```
2012-12-12T13:44:33.448 INFO [acsStart] Starting ACS version 10.2.0
2012-12-12T13:44:33.455 INFO [acsStart] Using ALMA Base Software tag WARNING: No ACS_TAG available
2012-12-12T13:44:33.542 INFO [acsStartORBSRVC] Starting CORBA Services
2012-12-12T13:44:33.554 INFO [acsStartORBSRVC]     Orb options:  -ORBDottedDecimalAddresses 1
2012-12-12T13:44:33.722 INFO [acsNamingService] Starting Naming Service
2012-12-12T13:44:34.924 INFO [acsInterfaceRepository] Starting Interface Repository
```

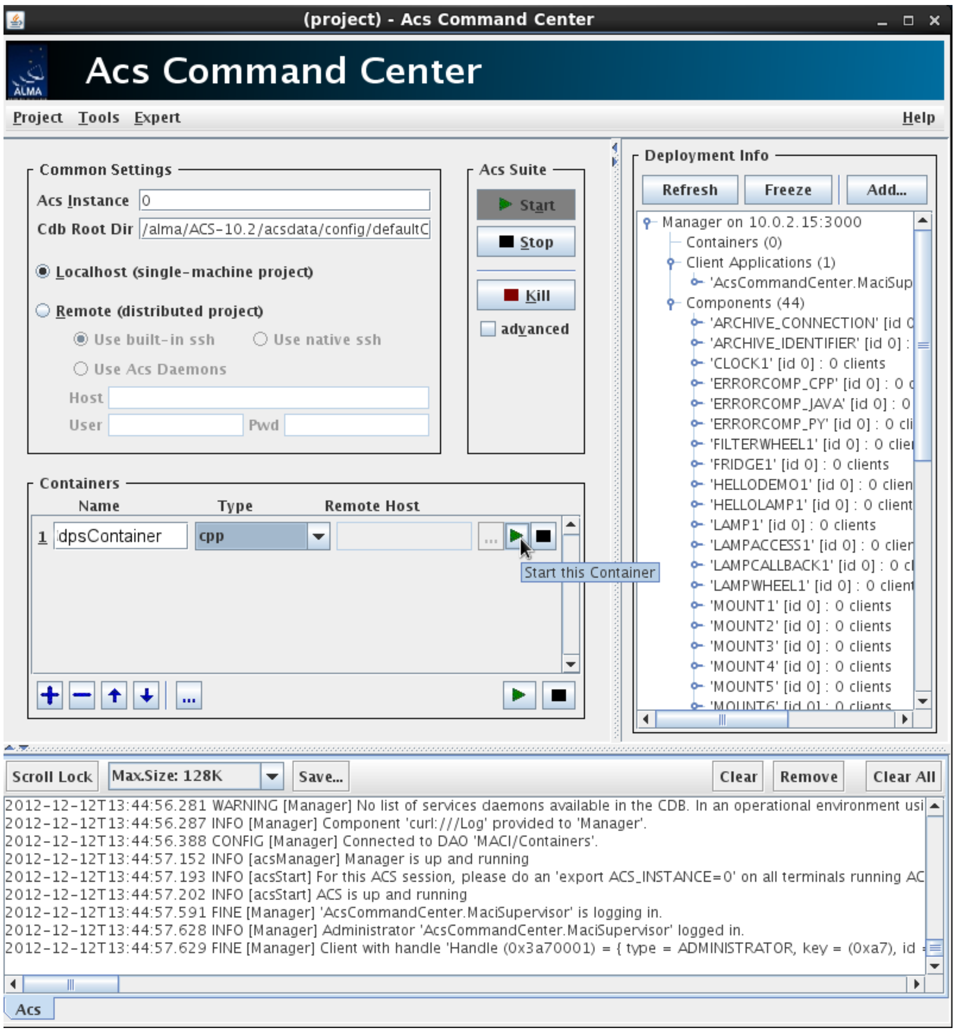
Figure 3 - Starting of ACS

When ACS starts the log panel (in the bottom box) is hidden. It can be extensive dragging over the mouse.

When the ACS is started, the following Container fields must set:

- Name of the container;
- Programming Language;

Then the container can be started by selecting the green arrow as shown in the following image:



Acs Instance: 0
Cdb Root Dir: /alma/ACS-10.2/acsdata/config/defaultC
(● Localhost (single-machine project)) ○ Remote (distributed project)
(● Use built-in ssh) ○ Use native ssh ○ Use Acs Daemons
Host:
User: Pwd:

1 dpsContainer cpp

```
Manager on 10.0.2.15:3000
  Containers (0)
  Client Applications (1)
    'AcsCommandCenter.MaciSup
  Components (44)
    'ARCHIVE_CONNECTION' [id 0
    'ARCHIVE_IDENTIFIER' [id 0] :
    'CLOCK1' [id 0] : 0 clients
    'ERRORCOMP_CPP' [id 0] : 0 c
    'ERRORCOMP_JAVA' [id 0] : 0
    'ERRORCOMP_PY' [id 0] : 0 cli
    'FILTERWHEEL1' [id 0] : 0 clie
    'FRIDGE1' [id 0] : 0 clients
    'HELLODEMO1' [id 0] : 0 clien
    'HELLOLAMP1' [id 0] : 0 client
    'LAMP1' [id 0] : 0 clients
    'LAMPACCESS1' [id 0] : 0 clier
    'LAMPCALLBACK1' [id 0] : 0 cl
    'LAMPWHEEL1' [id 0] : 0 client
    'MOUNT1' [id 0] : 0 clients
    'MOUNT2' [id 0] : 0 clients
    'MOUNT3' [id 0] : 0 clients
    'MOUNT4' [id 0] : 0 clients
    'MOUNT5' [id 0] : 0 clients
    'MOUNT6' [id 0] : 0 clients
```

Max.Size: 128K

```
2012-12-12T13:44:56.281 WARNING [Manager] No list of services daemons available in the CDB. In an operational environment usi
2012-12-12T13:44:56.287 INFO [Manager] Component 'curl:///Log' provided to 'Manager'.
2012-12-12T13:44:56.388 CONFIG [Manager] Connected to DAO 'MACI/Containers'.
2012-12-12T13:44:57.152 INFO [acsManager] Manager is up and running
2012-12-12T13:44:57.193 INFO [acsStart] For this ACS session, please do an 'export ACS_INSTANCE=0' on all terminals running AC
2012-12-12T13:44:57.202 INFO [acsStart] ACS is up and running
2012-12-12T13:44:57.591 FINE [Manager] 'AcsCommandCenter.MaciSupervisor' is logging in.
2012-12-12T13:44:57.628 INFO [Manager] Administrator 'AcsCommandCenter.MaciSupervisor' logged in.
2012-12-12T13:44:57.629 FINE [Manager] Client with handle 'Handle (0x3a70001) = { type = ADMINISTRATOR, key = (0xa7), id
```

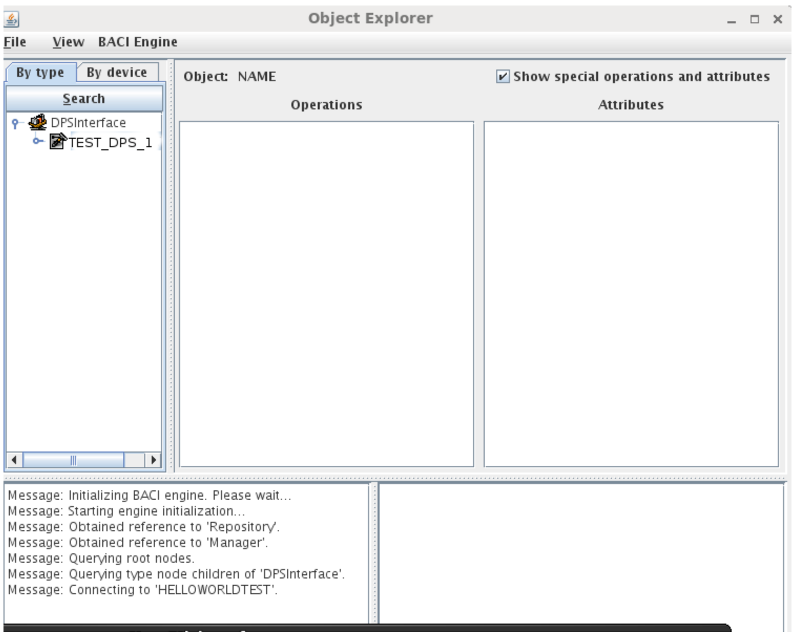
Figure 4 - Starting the Container

Now ACS is up and the Container of the tutorial is running as well. It is the moment to test the application just created. The ACS provide three useful interfaces:

1. Object explorer;
2. Log Panel;
3. Alarm Panel

14.1 Object explorer

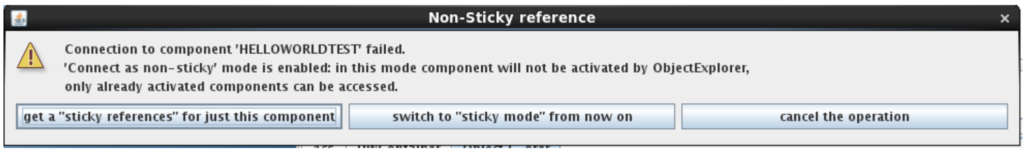
You can view and manipulate the component with the **Object Explorer** panel by selecting `Tools -> Object explorer` to show the following window:



```
Message: Initializing BACI engine. Please wait...
Message: Starting engine initialization...
Message: Obtained reference to 'Repository'.
Message: Obtained reference to 'Manager'.
Message: Querying root nodes.
Message: Querying type node children of 'DPSInterface'.
Message: Connecting to 'HELLOWORLDTEST'.
```

Figure 5 - Object Explorer

Click on TEST_DPS_1 to display the methods of the component. This will display this alert:

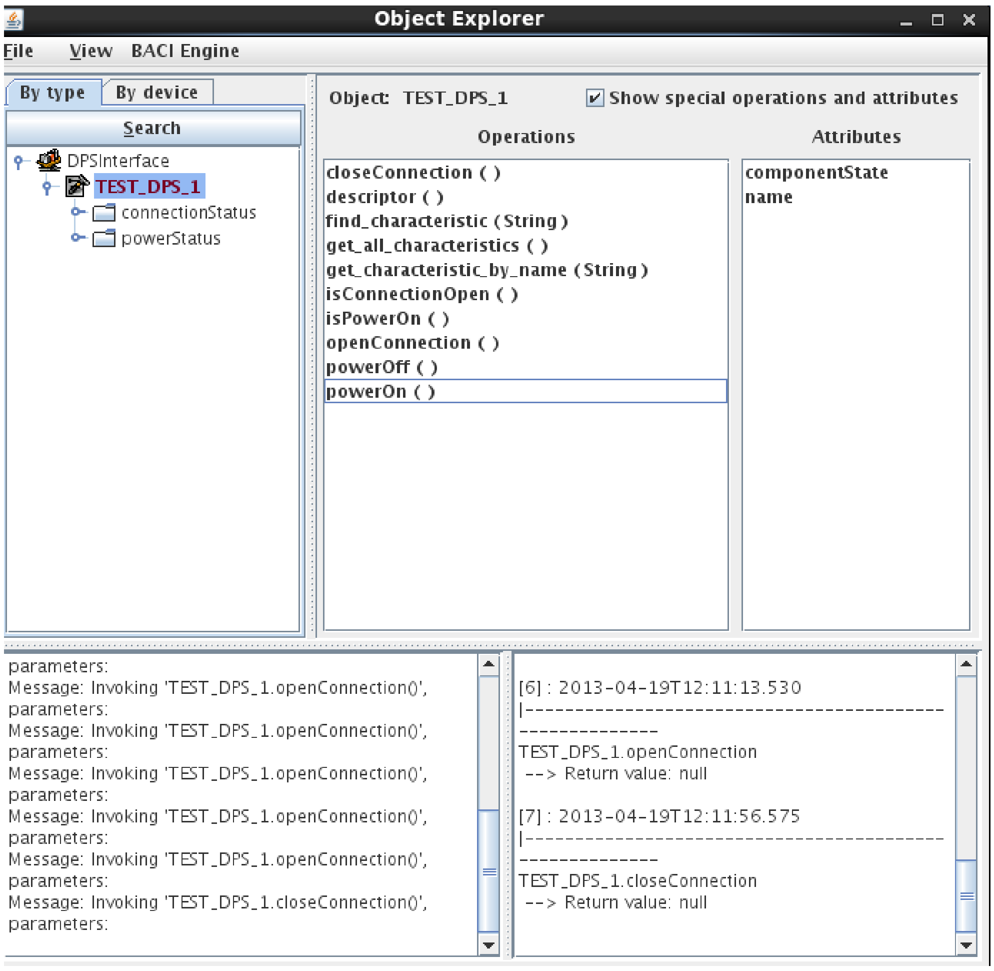

Non-Sticky reference
get a "sticky references" for just this component
switch to "sticky mode" from now on
cancel the operation

Connection to component 'HELLOWORLDTEST' failed.
'Connect as non-sticky' mode is enabled: in this mode component will not be activated by ObjectExplorer,
only already activated components can be accessed.

Figure 6 - Sticky Reference Error

! Any errors that appear at this stage need to be resolved before proceeding further. You should review all source code (especially XML files). The ACS logging client (jlog) can help you find the errors that occur at runtime. Jlog has many options for filtering on log fields, so you can suppress, for example, INFO logs in order to highlight ERROR and WARNING logs.

Click on the central button to continue and show the component methods:



```
closeConnection ( )
descriptor ( )
find_characteristic (String)
get_all_characteristics ( )
get_characteristic_by_name (String)
isConnectionOpen ( )
isPowerOn ( )
openConnection ( )
powerOff ( )
powerOn ( )
```

```
componentState
name
```

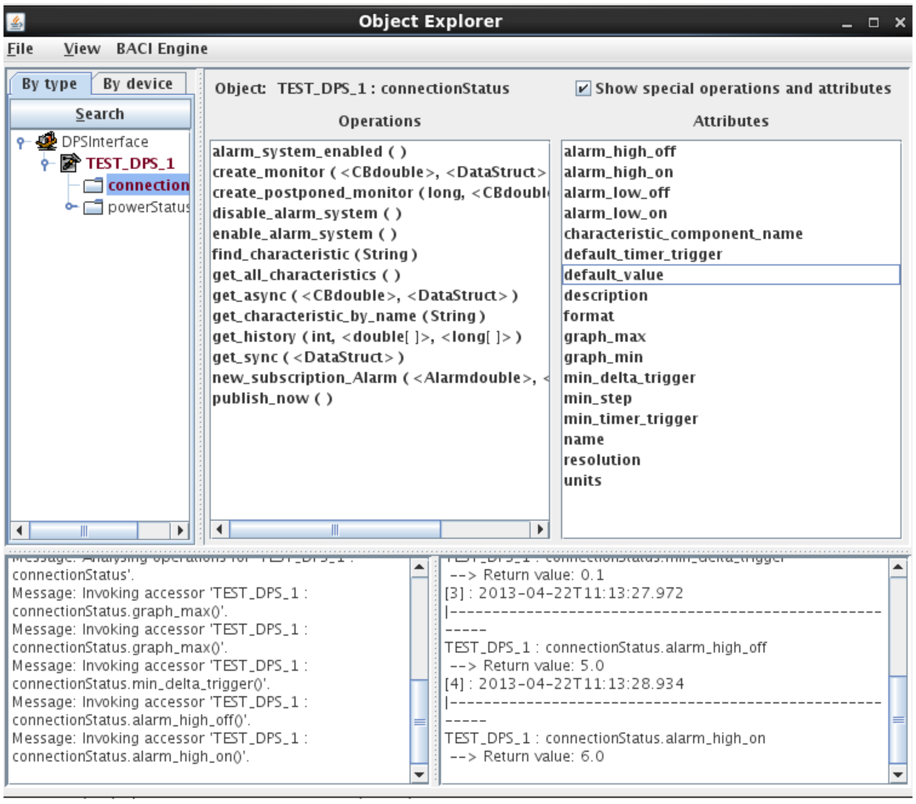
```
parameters:
Message: Invoking 'TEST_DPS_1.openConnection()',
parameters:
Message: Invoking 'TEST_DPS_1.openConnection()',
parameters:
Message: Invoking 'TEST_DPS_1.openConnection()',
parameters:
Message: Invoking 'TEST_DPS_1.openConnection()',
parameters:
Message: Invoking 'TEST_DPS_1.openConnection()',
parameters:
Message: Invoking 'TEST_DPS_1.closeConnection()',
parameters:
```

```
[6] : 2013-04-19T12:11:13.530
|------------------------------------------
--------------
TEST_DPS_1.openConnection
 --> Return value: null

[7] : 2013-04-19T12:11:56.575
|------------------------------------------
--------------
TEST_DPS_1.closeConnection
 --> Return value: null
```

Figure 7 - Component Methods

In the above image you can display all methods offered by the component and the two baci properties defined: *powerStatus* and *connectionsStatus* by expanding the TEST_DPS_1 icon. By selecting the one of the two BACI properties, you can see all its properties set in the xml file (description, default, value, alarms, ecc.) in the bottom-right of the following picture:



Object: TEST_DPS_1 : connectionStatus

Operations:

```
alarm_system_enabled ( )
create_monitor ( <CBdouble>, <DataStruct>
create_postponed_monitor ( long, <CBdoubl
disable_alarm_system ( )
enable_alarm_system ( )
find_characteristic ( String )
get_all_characteristics ( )
get_async ( <CBdouble>, <DataStruct> )
get_characteristic_by_name ( String )
get_history ( int, <double[ ]>, <long[ ]> )
get_sync ( <DataStruct> )
new_subscription_Alarm ( <Alarmdouble>, <
publish_now ( )
```

Attributes:

```
alarm_high_off
alarm_high_on
alarm_low_off
alarm_low_on
characteristic_component_name
default_timer_trigger
default_value
description
format
graph_max
graph_min
min_delta_trigger
min_step
min_timer_trigger
name
resolution
units
```

```
connectionStatus'.
Message: Invoking accessor 'TEST_DPS_1 :
connectionStatus.graph_max()'.
Message: Invoking accessor 'TEST_DPS_1 :
connectionStatus.graph_max()'.
Message: Invoking accessor 'TEST_DPS_1 :
connectionStatus.min_delta_trigger()'.
Message: Invoking accessor 'TEST_DPS_1 :
connectionStatus.alarm_high_off()'.
Message: Invoking accessor 'TEST_DPS_1 :
connectionStatus.alarm_high_on()'.
```

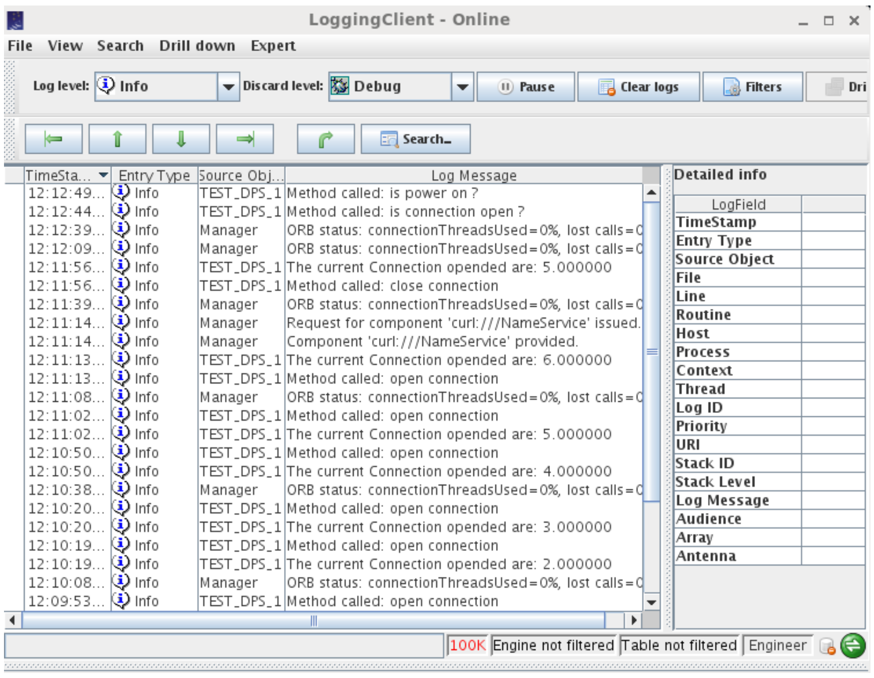
```
 --> Return value: 0.1
[3] : 2013-04-22T11:13:27.972
|----------------------------------------------------
-----
TEST_DPS_1 : connectionStatus.alarm_high_off
 --> Return value: 5.0
[4] : 2013-04-22T11:13:28.934
|----------------------------------------------------
-----
TEST_DPS_1 : connectionStatus.alarm_high_on
 --> Return value: 6.0
```

Figure 8 BACI properties

The object explorer is utils also to check the returned value of a function. Indeed in the rigth bottom of the Figure 8 is display the call methods name, the time of calling and the return value.

14.2 Logging Panel

The logging panel is open selecting Tool->logging client (graphical). It display all log generated both the ACS manager and the component just created. An example is shown in figure 9.



LoggingClient - Online

File View Search Drill down Expert

Log level: Info   Discard level: Debug   Pause   Clear logs   Filters   Dri

Search...

TimeSta...	Entry Type	Source Obj...	Log Message
12:12:49...	Info	TEST_DPS_1	Method called: is power on ?
12:12:44...	Info	TEST_DPS_1	Method called: is connection open ?
12:12:39...	Info	Manager	ORB status: connectionThreadsUsed=0%, lost calls=0
12:12:09...	Info	Manager	ORB status: connectionThreadsUsed=0%, lost calls=0
12:11:56...	Info	TEST_DPS_1	The current Connection opended are: 5.000000
12:11:56...	Info	TEST_DPS_1	Method called: close connection
12:11:39...	Info	Manager	ORB status: connectionThreadsUsed=0%, lost calls=0
12:11:14...	Info	Manager	Request for component 'curl:///NameService' issued.
12:11:14...	Info	Manager	Component 'curl:///NameService' provided.
12:11:13...	Info	TEST_DPS_1	The current Connection opended are: 6.000000
12:11:13...	Info	TEST_DPS_1	Method called: open connection
12:11:08...	Info	Manager	ORB status: connectionThreadsUsed=0%, lost calls=0
12:11:02...	Info	TEST_DPS_1	Method called: open connection
12:11:02...	Info	TEST_DPS_1	The current Connection opended are: 5.000000
12:10:50...	Info	TEST_DPS_1	Method called: open connection
12:10:50...	Info	TEST_DPS_1	The current Connection opended are: 4.000000
12:10:38...	Info	Manager	ORB status: connectionThreadsUsed=0%, lost calls=0
12:10:20...	Info	TEST_DPS_1	Method called: open connection
12:10:20...	Info	TEST_DPS_1	The current Connection opended are: 3.000000
12:10:19...	Info	TEST_DPS_1	Method called: open connection
12:10:19...	Info	TEST_DPS_1	The current Connection opended are: 2.000000
12:10:08...	Info	Manager	ORB status: connectionThreadsUsed=0%, lost calls=0
12:09:53...	Info	TEST_DPS_1	Method called: open connection

Detailed info

LogField	
TimeStamp	
Entry Type	
Source Object	
File	
Line	
Routine	
Host	
Process	
Context	
Thread	
Log ID	
Priority	
URI	
Stack ID	
Stack Level	
Log Message	
Audience	
Array	
Antenna	

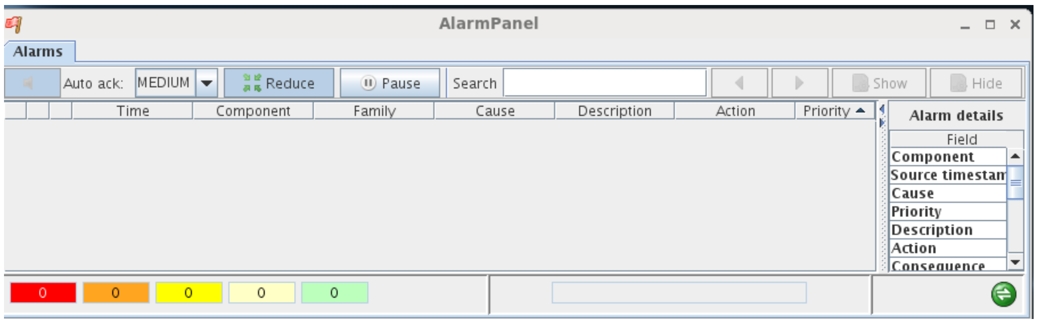
100K | Engine not filtered | Table not filtered | Engineer

Figura 9 - Logging Panel

The logging panel table report the timestamp in which the log is created, the type, the source and the message. The log is very useful to monitor the component execution but it is most important do not exceed otherwise will be product too much data with many difficult to handle.

14.3 The Alarm Panel

The Alarm Panel must be executed by a terminal (is required that ACS is just running) with the command: `alarmPanel`. The Figure 10 display the alarmPanel just run.



AlarmPanel

Alarms

Auto ack: MEDIUM   Reduce   Pause   Search   Show   Hide

			Time	Component	Family	Cause	Description	Action	Priority

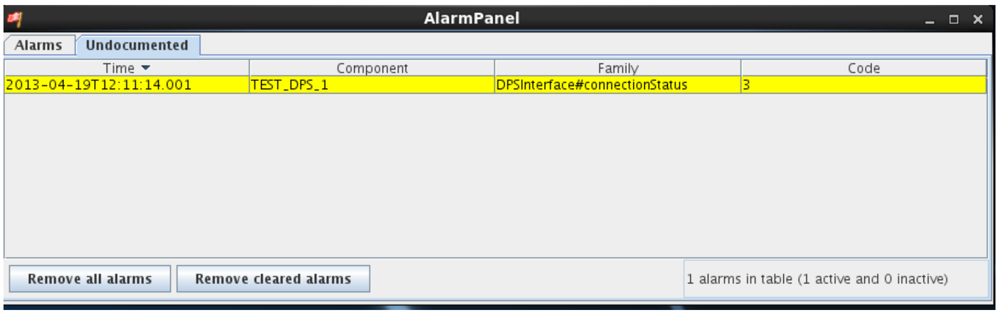
Alarm details

Field	
Component	
Source timestam	
Cause	
Priority	
Description	
Action	
Consequence	

🟥 0 🟧 0 🟨 0 0 🟩 0

Figura 10 - Alarm panel

When it is created an alarm, for example because the number of the connections opened is out of the allowed range, the alarm Panel display soon a new row as in the following figure:



AlarmPanel

Alarms | Undocumented

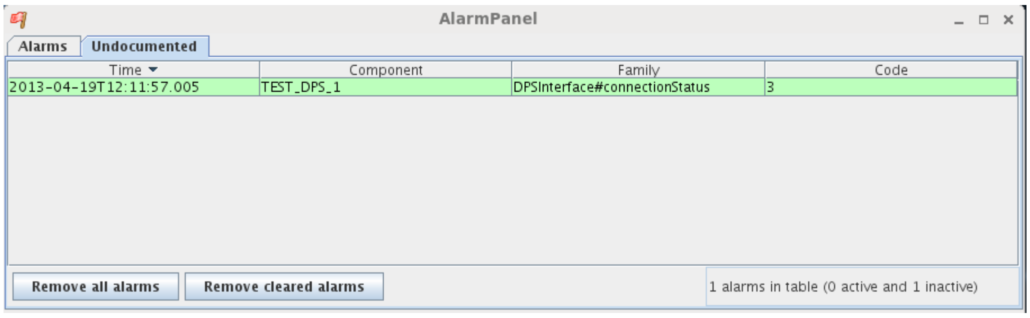
Time	Component	Family	Code
2013-04-19T12:11:14.001	TEST_DPS_1	DPSInterface#connectionStatus	3

Remove all alarms | Remove cleared alarms

1 alarms in table (1 active and 0 inactive)

Figura 11 New alarm

If the alarm is finished (for example because the operator has cloned some connection) then it is notify on the alarm Panel as in the figure 12.



AlarmPanel

Alarms | Undocumented

Time	Component	Family	Code
2013-04-19T12:11:57.005	TEST_DPS_1	DPSInterface#connectionStatus	3

Remove all alarms | Remove cleared alarms

1 alarms in table (0 active and 1 inactive)

Figura 12 Alarm ended

The operator can delete everytime the ended alarm rows.

15. REPOSITORY

When you use the ACS framework, and especially when you’re working in a team you should maintain yoursource code in a code repository. A solution just proposed in ASTRI project is GIT:

`http://redmine.iasfbo.inaf.it/projects/astri/wiki/ASTRI_sw_git_server_at_IASFBO`

16. DELIVERABLES

17. CONTACTS

Vito Conforti

E-mail:	conforti@iasfbo.inaf.it
Phone:	+39.051.639.8735

Andrea Bulgarelli

E-mail:	bulgarelli@iasfbo.inaf.it
Phone:	+39.051.639.8774

Massimo Trifoglio

E-mail:	trifoglio@iasfbo.inaf.it
Phone:	+39.051.639.8738

Fulvio Gianotti

E-mail:	gianotti@iasfbo.inaf.it
Phone:	+39.051.639.8706

18. ANNEX A

18.1 AlarmSystemConfiguration.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<alarm-system-configuration
   xmlns="urn:schemas-cosylab-com:acsalarm-alarmservice:1.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

        <configuration-property name="Implementation">CERN</configuration-property>
</alarm-system-configuration>
```

18.2 Categories.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<categories
        xmlns="urn:schemas-cosylab-com:acsalarm-categories:1.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <category is-default="true" path="CATEGORY1">
    <description>Test category 1</description>
    <alarms>
      <FaultFamily>BaciPropTest#testDoubleVar</FaultFamily>
      <FaultFamily>BaciPropTest#testPatternVar</FaultFamily>
      <FaultFamily>TestFF</FaultFamily>
      <FaultFamily>AnotherFF</FaultFamily>
    </alarms>
  </category>
</categories>
```

18.3 ReductionDefinition.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
  - Sample configuration of alarm reduction links.
 -->
<reduction-definitions
   xmlns="urn:schemas-cosylab-com:AcsAlarmSystem:1.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<!--
        <links-to-create/>

        <thresholds/>
-->

</reduction-definitions>
```

18.4 BACIProperty.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<fault-family name="BACIProperty"
        xmlns="urn:schemas-cosylab-com:acsalarm-fault-family:1.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <alarm-source>ALARM_SYSTEM_SOURCES</alarm-source>
  <help-url>http://tempuri.org</help-url>
  <contact name="Test"/>
  <fault-code value="1">
    <priority>1</priority>
    <problem-description>BACI property</problem-description>
  </fault-code>
  <fault-code value="2">
    <priority>1</priority>
    <problem-description>BACI property (LOW)</problem-description>
  </fault-code>
  <fault-code value="3">
    <priority>1</priority>
    <problem-description>BACI property (HIGH)</problem-description>
  </fault-code>
  <fault-member-default>
  </fault-member-default>
</fault-family>
```

18.5 BaciPropTest#testDoubleVar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<fault-family name="BaciPropTest#testDoubleVar"
        xmlns="urn:schemas-cosylab-com:acsalarm-fault-family:1.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <alarm-source>ALARM_SYSTEM_SOURCES</alarm-source>
  <help-url>http://tempuri.org</help-url>
  <contact name="Test"/>
  <fault-code value="1">
    <priority>1</priority>
    <problem-description>BACI property</problem-description>
  </fault-code>
  <fault-code value="2">
    <priority>1</priority>
    <problem-description>BACI property (LOW)</problem-description>
  </fault-code>
  <fault-code value="3">
    <priority>1</priority>
    <problem-description>BACI property (HIGH)</problem-description>
  </fault-code>
  <fault-member-default>
  </fault-member-default>
</fault-family>
```

18.6 DPSInterface.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<fault-family name="DPSInterface"
       xmlns="urn:schemas-cosylab-com:acsalarm-fault-family:1.0"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <alarm-source>ALARM_SYSTEM_SOURCES</alarm-source>
  <help-url>http://tempuri.org</help-url>
       <contact name="Vito" />
  <fault-code value="1">
    <priority>1</priority>
    <problem-description>BACI property with aanother FF, FM</problem-description>
  </fault-code>
  <fault-code value="2">
    <priority>1</priority>
    <problem-description>BACI property with a another FF, FM (LOW)</problem-description>
  </fault-code>
  <fault-code value="3">
    <priority>1</priority>
    <problem-description>BACI property with a another FF, FM (HIGH)</problem-
description>
  </fault-code>
  <fault-member-default>
  </fault-member-default>
</fault-family>
```

18.7 Manager.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<fault-family name="Manager" xmlns="urn:schemas-cosylab-com:acsalarm-fault-family:1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <alarm-source>ALARM_SYSTEM_SOURCES</alarm-source>
  <help-url>http://tempuri.org</help-url>
  <contact name="Alessandro Caproni"/>

  <fault-code value="1">
    <priority>3</priority>
    <problem-description>Container crashed</problem-description>
  </fault-code>

  <fault-code value="2">
    <priority>2</priority>
    <problem-description>Filesystem error affecting manager state recovery after
restart.</problem-description>
  </fault-code>

  <!--  Having a default fault member is necessary for alarms on container/client
crashes (FC=1),
        because for these the manager uses FM=<clientName> which cannot be configured
statically.
   -->
  <fault-member-default/>

  <fault-member name="Prevayler"/>

</fault-family>
```

18.8 TestFF.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<fault-family name="TestFF"
       xmlns="urn:schemas-cosylab-com:acsalarm-fault-family:1.0"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <alarm-source>ALARM_SYSTEM_SOURCES</alarm-source>
  <help-url>http://tempuri.org</help-url>
  <contact name="ACS developer"/>
  <fault-code value="1">
    <priority>1</priority>
    <problem-description>BACI property with a new FF, FM</problem-description>
  </fault-code>
  <fault-code value="2">
    <priority>1</priority>
    <problem-description>BACI property with a new FF, FM (LOW)</problem-description>
  </fault-code>
  <fault-code value="3">
    <priority>1</priority>
    <problem-description>BACI property with a new FF, FM (HIGH)</problem-description>
  </fault-code>
  <fault-member-default>
  </fault-member-default>
</fault-family>
```