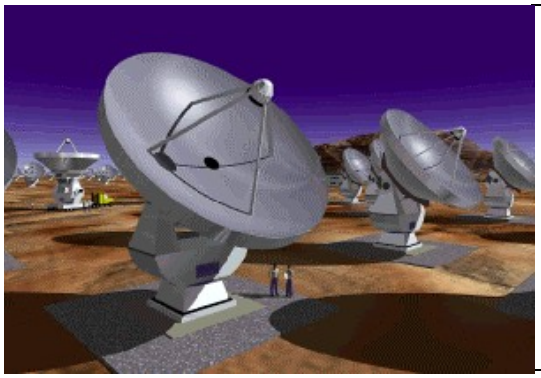

Atacama Large Millimeter	ALMA-SW-NNNN Revision: 4.0 2004-11-09

ALMA Common Software
Time System

Design and Tutorial

David Fugate
University of Calgary

Keywords: ACS, Time, OMG, Epoch, Duration, C++, Java, Python, TAI, UTC, Array Time, Clock, Timer, TimeoutHandler, timeService, TICS

Author Signature: David Fugate	Date: 3-8-2004
Approved by:	Signature:
Institute:	Date:
Released by:	Signature:
Institute:	Date:

Change Record

REVISION	DATE	AUTHOR	SECTIONS/PAGES AFFECTED
	REMARKS		
3.0	2004-03-08	D. Fugate	All
	Created rough draft.		
3.1.0	2004-04-22	D. Fugate	All
	Updated for 3.1		
3.1.0.1	2004-04-29	D. Fugate	All
	Improved formatting and documented newly introduced 3.1 changes.		
4.0.0	2004-11-09	D. Fugate	All
	Updated for 4.0		

Table of Contents

1 Overview

The ALMA Common Software Time System, based on code from the *TICS timeService* module, was designed to provide the following functionality to ALMA developers:

- Obtaining the current array time in the standard OMG time format
- Converting array time to various other representations including but not limited to ISO-8601, Universal Coordinated Time, etc.
- The ability to set "timeouts" which asynchronously invoke developer code at a specific time and/or frequency.
- Convert native language time representations to and from OMG array time.

What's listed above has been accomplished through the implementation of two standard ACS components: *Clock* and *Timer* respectively. While this document does not cover all possible uses of these components (the Doxygen-generated documentation for *acstime.idl* should be more than sufficient for this purpose), it does provide sample implementations of the IDL *TimeoutHandler* interface in all three CORBA implementation languages supported by ACS. It would be of great benefit to the developer to refer to the glossary of this document constantly as some of the terms used in the Time System can be quite confusing.

It's essential to note that the ACS Time System does **not** include support for the 48ms clock based in the antennas' hardware. While there were specific IDL methods dealing

with this pulse in the *TICS timeService* module, these have since been removed as it was determined knowledge of the timing event was irrelevant outside of the Control and Correlator ALMA subsystems.

1.1 Glossary

ACS ALMA Common Software

Array Time The synchronized time according to all ALMA antennas???

Clock An IDL interface implemented as a standard component which gives users the current OMG array time. Additionally, *Clock* provides a few time conversion functions along with public access to the variations between array, TAI, and UTC times.

Duration The difference between two epochs. This is a signed value and in IDL it's represented by the *ACS::TimeInterval* typedef (i.e., *long long*) embedded inside a struct.

Epoch A unique instance in (OMG) time. In IDL, an epoch unit is represented by the typedef *ACS::Time* (i.e., *unsigned long long*) embedded within a struct.

IDL Interface Definition Language

OMG Time	The format of time used by the Object Management Group is based on the beginning of the Gregorian calendar. This is strictly defined as the number of one-hundred nanosecond units that have occurred since October 15, 1582 at precisely 00:00:00. For reference, the last representable date using this format is March 11, 60038.
TAI	TAI is short for International Atomic Time. Based on the second unit and derived from a large number of clocks all over the world.
TimeoutHandler	An IDL interface which **must be implemented by the developer** to be used. *TimeoutHandler* defines a "handleTimeout" method which shall only be invoked by *Timer* components.
TICS	Test Interferometer Control Software. Now known as the ALMA Control Software development group.
Timer	An IDL interface implemented as a standard component which is capable of executing developer code at a given instance in time (at a specific frequency if desired). The code to be invoked is passed to the *Timer* component via the implementation of a *TimeoutHandler* interface.
timeService	A software module that was located in the old ALMA CMM archive and was the basis for the *acstime* software module.
UTC	UTC is short for Coordinated Universal Time. Based on the second unit but unlike TAI, UTC keeps track of so-called leap seconds. Offset from TAI is always an integer number of seconds.

1.2 References

[R01] ACS Architecture 4.0

(http://www.eso.org/~almamgr/AlmaAcs/OnlineDocs/ACSArchitecture-4.0.pdf)

[R02] ACS Software Module(s): "acstime", "acspy", "jacsutil"

[R03] CORBA

(http://www.omg.org/technology/documents/formal/corba_iiop.htm)

2 ACS 3.1 to 4.0 Porting Guide

In an effort to reduce duplicated documentation throughout ACS which can lead to inconsistencies, developers should now look at the ACS 4.0 Release Notes for information regarding required and suggested changes: http://almasw.hq.eso.org/almasw/bin/view/ACS/ReleaseNotes_ACS_4_0_0

Essentially all you'll find there is that a few additional constructors supporting seconds (long double format) have been added to the C++ *DurationHelper* and *EpochHelper* classes. Also, a new Java class found in the *alma.acs.time.TimeHelper* package has been added providing additional time conversion utilities.

3 Design

3.1 Requirements

Please refer to section 3.9 of the ALMA Common Software Architecture document (**[R03]**).

3.2 acstime.idl

Complete descriptions of all data types and interfaces used within the ACS Time System can be found in the Doxygen-generated documentation for the *acsclock* IDL module. That being said, very brief, high-level overviews of the standard ACS Time System components are documented below.

Interface	Description
Clock	An interface used to obtain the current time and provide some general time conversion functionality. *Clock* is normally only used as a collocated object and will normally be present on all machines (i.e. on both LCUs and general-purpose workstations).
TimeoutHandler	Create an object of this class to receive timeout events (not to be confused with channel events). This interface provides a callback type mechanism and can really be thought of as an alarm. A reference to this CORBA object is passed as an argument to *Timer*'s schedule method. Other arguments to schedule are when the timeout(s) will occur and how often.
Timer	An interface designed to schedule timeouts for *TimeoutHandler* instances. These timeouts can be set to occur only once or on some given frequency. The only other interesting thing to note here is timeouts can be cancelled.

4 Time Helper Classes

C++

There are three classes available in C++ - *TimeUtil*, *EpochHelper*, and *DurationHelper*. *TimeUtil* converts ACS Epoch and Duration structures to and from ACE time formats. Both *EpochHelper* and *DurationHelper* are wrapper classes for the IDL Epoch and Duration structs which overload various operators and provide string conversion methods. More detailed information on these classes and the exact methods they provide can be found in the Doxygen-generated documentation.

Java

There are two helper classes provided in Java – *alma.acs.util.UTCUtility* and *alma.acs.time.TimeHelper*. They both provide time conversion functionality. Please see the Doxygen-generated documentation for specific usage.

Python

Python support is nearly identical to C++. The Acspy.*Common.TimeHelper* package provides *TimeUtil*, *EpochHelper*, and *DurationHelper* classes as well as a *getTimeStamp* method. *EpochHelper* and *DurationHelper* are actually generated from the C++ classes of the same name using SWIG so the C++ Doxygen documentation applies here as well. *TimeUtil* only differs from the C++ implementation in the fact that it converts OMG time to and from native Python time format(s). For precise information on the functionality provided by *TimeUtil*, please see the pydoc.

5 TimeoutHandler Implementations

Nearly everything in the ACS Time System is provided as a service so there is hardly anything for the developer to implement. In simple terms, the Time System API includes implemented components where developers need only invoke methods to obtain the desired results. The exception to this is that developers must provide an implementation of the *acstime::TimeoutHandler* IDL interface if they want a certain code segment executed at a given time and/or frequency. The examples listed below show exactly how to do this in each implementation language supported by ACS.

C++

This example provides the full implementation of a *TimeoutHandler* C++ class. Detailed information on retrieving a reference to a *Timer* component has been omitted as this is covered in many other ACS documents and examples. Please note that **bolded** code should be adapted for your particular needs.

```
1.  #include "acstime.h"
2.
3.  class TimeoutHandlerImpl : public virtual POA_acstime::TimeoutHandler
4.  {
5.    public:
6.      void handleTimeout(const acstime::Epoch& e)
7.        throw (CORBA::SystemException)
8.      {
9.        cout << "The current time is:" << e.value << endl;
10.     }
11. };
12.
13. //Obtain a reference to a Timer component named “joe”
14.
15. EpochHelper *epoch_p = new EpochHelper();
16. DurationHelper *duration_p = new DurationHelper();
17.
18. acstime::Epoch start;
19. start.value = m_now->get_sync(completion.out());
20. epoch_p->value(start);
21. duration_p->second(3);
22. epoch_p->add(duration_p->value());
23.
```

```
24. duration_p->second(1);
25.
26. TimeoutHandlerImpl* mh1_p = new TimeoutHandlerImpl();
27. acstime::TimeoutHandler_ptr corbaTH_p = mh1_p->_this();
28.
29. int id1 = joe->schedule(corbaTH_p, epoch_p->value(),  duration_p->value());
```

1 The standard include for the ACS Time System provides access to the CORBA client stubs defined in *acstime.idl* as well as all of the helper classes.

3-11 An implementation of the *TimeoutHandler* interface must be defined. It is impossible to schedule timeouts without it.

6-10 Implementation of the IDL "handleTimeout" method. When the timeout occurs, this method will be invoked asynchronously by the ACS Time System.

6 The single parameter is the current time in the form of an Epoch struct. Really the developer only cares about this value when there are periodic timeouts on some interval specified to the *Timer* component.

9 In this simple example, we just print the current OMG (array) time to standard out.

13 If you do not understand how to retrieve references to other components, please review the acsexmpl module.

15-16 Create a couple of ACS helper objects to manipulate Epoch and Duration structs.

18-22 Determine exactly when 3 seconds from *TimeUtil* will be in OMG time.

19 The code used to obtain a reference to the *Clock* interface's *TimeUtil* property has been omitted as this is considered trivial material. Please see the BACI Device Server Programming Tutorial if needed.

24 Timeouts will occur once per second until they are cancelled.

26-27 Create the *TimeoutHandler* object and implicitly activate it as a CORBA object.

29 In a little under three seconds, line 9 will be printed to standard out once per second forever as a result of this statement. The timeouts could be cancelled using *Timer*'s cancel method with the id1 variable though.

Java

Essentially identical to the C++ example except that the *EpochHelper* and *DurationHelper* classes are not available.

```
1.  import alma.acstime.TimeoutHandlerHelper
2.
3.  private class TimeoutHandlerImpl extends alma.acstime.TimeoutHandlerPOA
4.  {
5.    public TimeoutHandlerImpl()
6.    { }
7.    public void handleTimeout(alma.acstime.Epoch e)
8.    {
9.       System.out.println("The current time is: " + e.value);
10.   }
11. }
12.
13. //Obtain a reference to a Timer component named “joe”
14. //…
15.
16. CompletionHolder completionHolder = new CompletionHolder();
17. alma.acstime.Epoch start = new
      alma.acstime.Epoch(m_now.get_sync(completionHolder));
18. start.value = start.value + 3000000;
19.
20. alma.acstime.Duration period = new alma.acstime.Duration(1000000);
21.
```

```
22. TimeoutHandlerImpl myHandler = new TimeoutHandlerImpl();
23. TimeoutHandler handlerCORBARef =
24.         TimeoutHandlerHelper.narrow(getContainerServices().activateOffShoot(myHandler)
    );
25.
26. int id1 = joe.schedule(handlerCORBARef, start, period);
```

1 Import a useful CORBA stub.

3-11 An implementation of the *TimeoutHandler* interface must be defined. It is impossible to schedule timeouts without it.

7-10 Implementation of the IDL "handleTimeout" method. When the timeout occurs, this method will be invoked asynchronously by the ACS Time System.

7 The single parameter is the current time in the form of an Epoch struct.

9 In this simple example, we just print the current OMG (array) time to standard out.

13 If you do not understand how to retrieve references to other components, please review the jcontexmpl module.

16-18 Determine exactly when 3 seconds from *TimeUtil* will be in OMG time.

17 The code used to obtain a reference to the *Clock* interface's *TimeUtil* property has been omitted as this is considered trivial material and there are many examples on obtaining references to BACI properties.

20 Timeouts will occur once per second until they are cancelled.

22-24 Create the *TimeoutHandler* object and activate it as a CORBA object using the ContainerServices.

26 In a little under three seconds, line 9 will be printed to standard out once per second forever as a result of this line. The timeouts could be cancelled using *Timer*'s cancel method with the id1 variable though.

Python

Nearly identical to the C++ example except that the available *EpochHelper* and *DurationHelper* classes have been left out to keep things simple. If you were to utilize these classes in your code, usage would be identical to the C++ example.

```
1.  import acstime, acstime__POA
2.
3.  class TimeoutHandlerImpl(acstime__POA.TimeoutHandler):
```

```
4.      def handleTimeout(self, e):
5.          print "The current time is: ", e.value
6.
7.  #Obtain a reference to a Timer component named “joe”
8.
9.  start = long(m_now().get_sync()[0]) + long(30000000)
10. start = acstime.Epoch(start)
11.
12. period = acstime.Duration(long(10000000))
13.
14. myHandler = TimeoutHandlerImpl()
15.
16. id1 = joe.schedule(myHandler._this(), start, period)
```

1 Import useful CORBA stubs.

3-5 An implementation of the *TimeoutHandler* interface must be defined. It is impossible to schedule timeouts without it.

4-5 Implementation of the IDL “handleTimeout” method. When the timeout occurs, this method will be invoked asynchronously by the ACS Time System.

4 The single parameter is the current time in the form of an Epoch struct.

5 In this simple example, we just print the current OMG (array) time to standard out.

7 If you do not understand how to retrieve references to other components, please review the acspyexmpl module.

9-10 Determine exactly when 3 seconds from *TimeUtil* will be in OMG time.

9 The code used to obtain a reference to the *Clock* interface’s *TimeUtil* property has been omitted as this is considered trivial material and there are many examples on obtaining references to BACI properties.

12 Timeouts will occur once per second until they are cancelled.

14-16 Create the *TimeoutHandler* object and implicitly activate it as a CORBA object using the *_this* method (line 16).

16 In a little under three seconds, line 5 will be printed to standard out once per second forever as a result of this line. The timeouts could be cancelled using *Timer*’s cancel method with the id1 variable though.

6 Other Useful Information

Editing the Configuration Database

Please simply copy the TIMER1 and CLOCK1 entries in $ACS_CDB/CDB/MACI/Components/Components.xml to your own CDB if access to the ACS Time Service is needed. Don't forget to also copy $ACS_CDB/CDB/alma/CLOCK1.

Makefile Support

In Python and Java, there is nothing special that must be done. For C++ though, developers must link in the *acstime* shared library.

7 Known Problems and Issues

Java helper classes have a lot missing compared to the C++ and Python implementations.

Subsystems which primarily use Java for their development have different needs for the Time System when compared to Control-oriented software which uses C++/Python to a great extent. When a specific need arises, please submit an SPR and the desired features will be implemented as soon as humanly possible.

8 Appendix

Time System Examples in the ALMA CVS Repository

ACS/LGPL/CommonSoftware/acstime/ws/test/ (C++)

ACS/LGPL/CommonSoftware/jcontexmpl/src/alma/demo/client/TimeoutHandlerClient

ACS/LGPL/CommonSoftware/acspyexmpl/src/acspyexmplTimeoutHandler

acstime.idl (ALMA CVS Repository)

ACS/LGPL/CommonSoftware/acstime/ws/idl/acstime.idl