Created:July 20, 1994
Modified:January 15, 1997

Gemini Controls Group Interface Control Document

ICD 1a
The System Command Interface

Kim Gillies and Steve Wampler

gscg.kkg.009/013

This report describes the software interface between OCS applications and other principal systems.

1.0 Introduction

1.1 Purpose

This ICD document specifies the principal system software interface that lies between and directly couples two communicating principal systems.

In detail it:

1. Indicates the nature and behavior of the software interface.
2. The functionality and software interface principal systems work package developers can expect from the OCS group software.
3. The functionality and software interface the OCS work package developers can expect from the other principal systems developers.

1.2 Scope

The ICD1 documents describe the OCS command architecture and define the requirements for the software interface between the Command Layer and the Attribute/Value Layer in the Systems Interface [3]. This document describes the software that will be provided by the Observatory Control System group that allows one principal system to command another principal system or another OCS application. This document together with the other ICD1X (where X can be b, c...) documents describe the entire command path from the top of the system to the hardware.

Details and specifications of the implementation of the Attribute/Value layer in an EPICS-based principal system are found in ICD1b [2]. Details and specifications of the implementation of the Attribute/Value layer in the Data Handling System is given in ICD1c.

For EPICS-based principal systems, no software need be written by work package groups to receive commands from the OCS or to post status. Therefore, this document now focuses on providing a general understanding of the software that is provided by the OCS and Gemini Project Office and how the software should be used by work package groups.

This document, ICD1a, provides only an overview of the OCS command architecture. The details of the OCS design and implementation appear in the OCS Work Project design documents. The most recently published OCS design information appears in the OCS Preliminary Design Documents [7]. This functionality is within the OCS work package and the detailed design of the OCS is not required to understand this ICD. This document has been changed to reflect the evolution of the OCS and system designs as those systems have been refined during the review process.

This document does not describe the status/alarm principal system interface. That interface is described in ICD 2.

1.3 Applicable Documents

[1] SPE-C-G0037, *Software Design Description*, Gemini 8m Telescope Project.

[2] GSCG.grp.024, *ICD 1b - The Baseline Attribute/Value Interface*, Gemini 8m Telescope Project.

[3] GSCG.kkg.005, *Baseline Major Systems Interface*, Version 003, 28/6/94, Gemini 8m Telescope Project.

[4] GSCG.kkg.002, *OCS Behavioral Interface Model Report*, Version 001, 23/6/94, Gemini 8m Telescope Project.

[5] GSCG.grp.001, *A Report on the Interface Model Issues*, Gemini 8m Telescopes Project.

[6] GSCG.grp.016, *Glossary for the Gemini Software*, Gemini 8m Telescopes Project.

[7] OCS Preliminary Design Review Documents, (Not yet released by controls group), available from them. FIX THIS!

[8] GSCG.kkg.017, *Two Command Models,* Gemini 8m Telescopes Project.

[9] *EPICS: IOC Application Developers Guide,* Los Alamos National Laboratory.

[10] SPE-I-G0009, *Software Programming Standards,* Gemini 8m Telescopes Project.

[11] SPE-C-G023, *"The Gemini Standard Controller Hardware Documentation"*, Andrew Johnson, Royal Greenwich Observatory.

1.4 Abbreviations and Acronyms

ACM	Action Command Model
AVL	Attribute/Value Layer
CA	EPICS Channel Access message system

DHS	Data Handling System
EPICS	Experimental Physics and Industrial Control System
GCS	Gemini Control System
ICD	Interface Control Document
IOC	Input/Output Controller
OCS	Observatory Control System
PSBI	Principal Systems Baseline Interface
SDD	Software Design Description
TBP	To Be Provided

1.5 Glossary

Additional glossary information is available in [6].

Attribute — An attribute is a textual description of some part of a Gemini based hardware or software system. An attribute has an associated value.

Attribute Group — Attributes can be collected into related groups of zero or more.

Configuration — The set of attributes that detail the conditions necessary to move the system from one state to another is called a configuration.

Principal System — At the highest level in the GCS software decomposition, the software system is divided into four kinds of software systems called principal systems. The four types are called: the Data Handling System, the Observatory Control System, the Telescope Control System, and the Instrument Control System. There may be up to four concurrently executing Instrument Control Systems.

Service — A service is a software interprocess communication method. Services enable communication between two applications. An example of a service is EPICS Channel Access.

System Configuration — A configuration can be split into sets of attributes for each system. These sets are system configurations.

Systems Interface — The principal systems within the GCS software design interact with one another through the Systems Interface.

Value — The value is the data associated with a particular attribute.

1.6 Stylistic Conventions

Program text is shown in the `Courier` typeface.

2.0 Overview

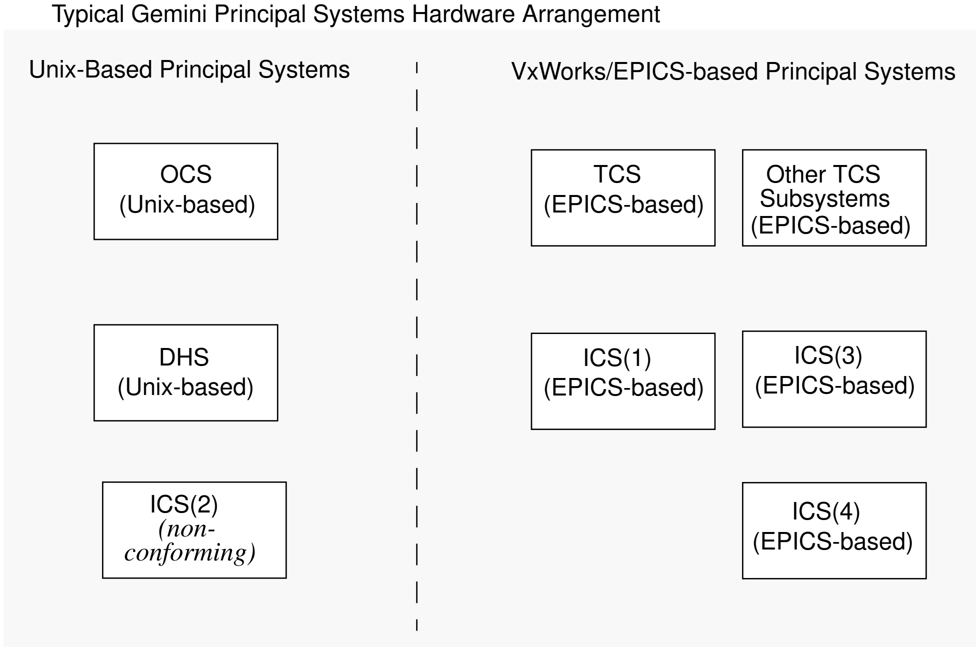
The Software Design Description [1], describes the Gemini Control System software. This document assumes familiarity with the software design and terms defined in the design document. Documents [3] and [4] can help relate the contents of this document to the overall software design.

2.1 System Hardware Architecture

The Gemini Control System is a distributed system executing on machines of different types and operating systems. The real-time principal systems software executing in the Telescope Control System and the facility Instrument Control Systems is based on a combination of VxWorks and EPICS. The Data Handling System and some visitor instruments may not be EPICS based, but the principal systems interface must allow communication between the OCS and all principal systems regardless of their operating environment.

Figure 1 shows how the principal systems would typically be distributed between VxWorks/EPICS systems and Unix-based systems. A “non-conforming” ICS is a facility ICS that is not based on EPICS/VxWorks.

FIGURE 1. Typical Gemini Principal Systems Hardware Arrangement



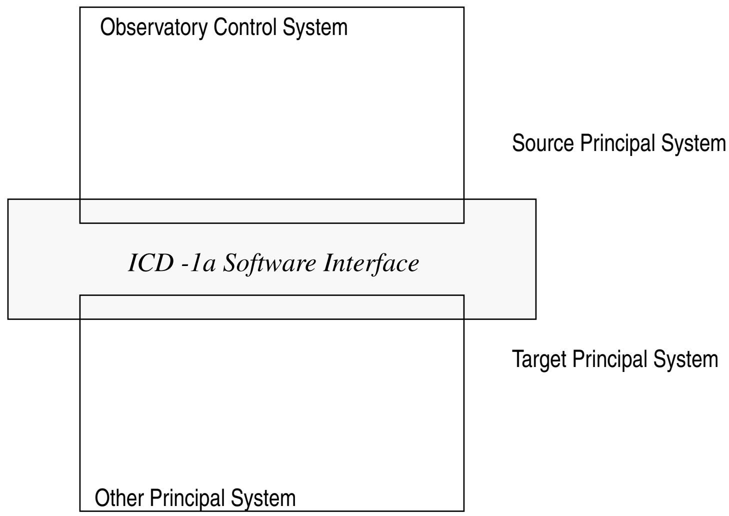
2.2 Communication Architecture

There is no hardware communication architecture specified. This interface describes the boundary between principal systems or Work Package groups.

2.2.1 Context Diagram

Figure 2 shows the relationship between ICD1a and the remaining software system.

FIGURE 2. ICD1 Context Diagram



2.2.2 Events and Responses

See Table 1 on page 8 and Table 2 on page 9 for the description of ICD1 Events and Responses.

3.0 Behavior

3.1 Design Overview

The OCS command design is based on the Principal System Baseline Interface (PSBI)[3], a layered design for communication. Figure 3 shows the PBSI. The Command Layer is above the Service Layer and below the Recipe and Console Command layers. The Recipe and Console Command Layer can use the features of the Command Layer or the Service Layer as is shown by the arrow descending from them to the Service Layer. This ability allows maximum flexibility.

The PSBI no longer depends on a Principal System Agent. The use of a Principal System Agent is an OCS implementation issue, not an interface issue (see later in this document).

This document focuses on the interface between two layers of the Principal Systems Baseline Interface (PSBI): the Service Layer and the Attribute/Value layer. The role of each of the layers follows. Note that this is a behavioral description and not an implementation description.

Recipe Layer. High-level OCS observing scripts manipulate configurations. The Recipe Layer provides script access to the configuration features of the Command Layer.

Console Command Layer. Some screens and scripts perform individual control commands. The Console Command Layer provides script support for the use of individual commands by applications.

Command Layer. The Command Layer builds upon the Service Layer features to provide configuration-based functionality to the Recipe and Console Command layers.

Service Layer. The Service Layer provides a service-independent software interface to the communication systems used in the software system. Each service does two things: it encapsulates an interprocess communication method and provides the "glue" that ties the Service Layer software interface to the service's Attribute/Value Layer.

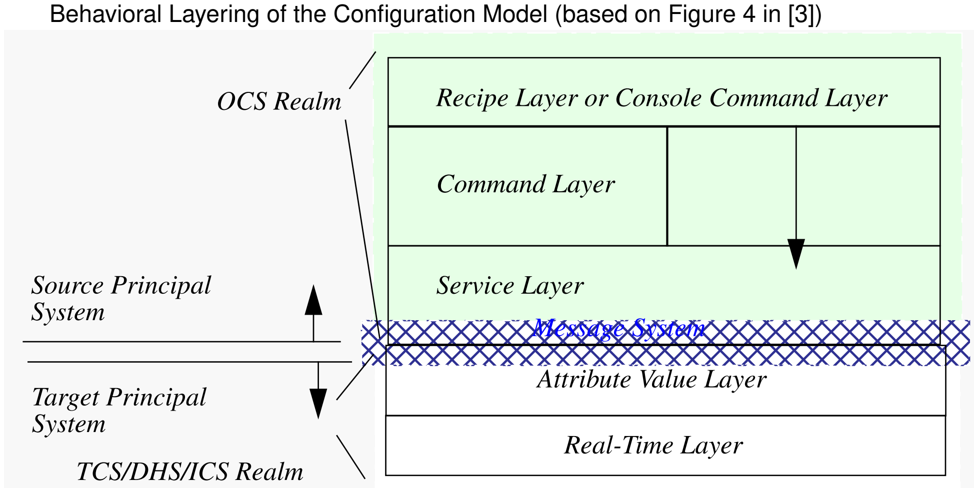
Attribute/Value Layer. This layer is the public software interface provided by a target application or principal system. This interface is used by the Service Layer to provide uniform behavior to the OCS.

Real-time Layer. The real-time layer contains whatever software is executing in a principal system or application. This layer provides the functionality of the system.

The following figure shows the software layering in the PSBI. The Recipe Layer, Console Command Layer, Command Layer, and Service Layer are entirely within the source principal system. The Attribute Value Layer is within the target principal system.

It is important to note that not every PSBI layer is required in all situations. The layers provide support for features in the OCS design but are not required in all situations. For instance, a console GUI can use just the EPICS Service through the Service Layer to send commands to an EPICS system without using the Command Layer features.

FIGURE 3. Behavioral Layering of the Configuration Model (based on Figure 4 in [3])



The cross-hatch pattern in Figure 3 shows the interface described in this document and specified in the related ICD1X documents (ICD1b and any other ICD1X documents that are needed in the future). The shaded portion above the cross-hatch pattern is software that is the responsibility of the OCS Work Package group.

The software below the pattern is the responsibility of the Gemini Controls Group in the Gemini Project Office. In particular, the EPICS attribute/value layer software is provided by this group.

It is the job of the implementation groups to use the Attribute/Value layer software properly (as described later in this document) and to document the commands that they provide to other systems. (SEE PDF???).

3.2 Command Layer

The Command Layer resides entirely within the OCS and is not visible to other systems. This layer accepts OCS commands from the Recipe Layer or the Console Command Layer. Each command includes a reference to a configuration.

The Command Layer separates a configuration into the system configurations for each destination principal system or application. It then routes each to the appropriate target using one of the Services provided by the Service Layer. Each Service converts its system configuration into a set of attribute/value pairs, along with a name of the OCS command to be executed.

The set of attribute/value pairs is communicated to the Attribute/Value Layer of the target system.

It is the Command Layer that is responsible for monitoring the progress of configuration commands and determining when each of the configuration commands has completed. Command completion is determined by examining the completion properties of each system configuration that is included in the command.

If directed, the Command Layer must notify the commanding layers above it when changes occur in the status of a command.

The Command Layer must monitor multiple, concurrent outstanding commands from multiple sources.

3.3 Service Layer

The OCS must provide an open, expandable command system. To meet this need the Service Layer provides a single, common software interface above whatever software is required to allow the OCS to communicate with another principal system or software component (called a Service).

The Service Layer provides the “glue” software that makes each Service behave in the manner described in the Service Layer software interface. Each Service knows about one type of Attribute/Value layer.

TABLE 1. Behavior of the Command Layer in Response to Events

Event	Response

Configuration command received	Return "Command received" acknowledgment.
	Register this as an outstanding configuration command.
	Split the command configuration into a set of system configurations.
	For each system configuration
	    Communicate the system configuration to the appropriate Service.
	    Ask to be notified when this system configuration is done or changed — do *not* wait for this.

"System configuration DONE" received	Determine which command this system configuration belonged to.
	Note the fact that this system configuration has been completed.
	Check for error.
	If all the system configurations for this command have been completed then
	    Synthesize and return "Configuration Command Completed".
	    Remove this command from the outstanding command register.
	End if

"System configuration changed" received	Determine which command this system configuration belonged to. Note the fact that this configuration has been changed. Report the change to whichever part of the system this command came from.

3.4 Attribute/Value Layer

This layer accepts lists of attribute/value pairs that have been communicated to the target principal system in a manner appropriate for the target system by a Service. Often, that may mean that the A/V pairs are grouped with others by the target system into attribute groups. The Service Layer must be aware of these groups and be able to signal the target system when an attribute group has been transmitted.

An Attribute/Value Layer has no knowledge of outstanding commands and should not retain command related state.

An Attribute/Value Layer must be capable of notifying its Service in the Service Layer when changes occur to attributes that have been set.

Proper operation of this layer requires that attributes exist in the target principal system or application for the status (Busy, Errors, etc.) of actions that result from attempting to match actual system attributes to commanded values.

TABLE 2. Behavior of Attribute/Value Layer in Response to Events

Event	Response
Attribute/Value pair or Attribute/Value group received from source Principal System	Return “A/V received” acknowledgment Process the A/V pair or group as needed for the principal system.
“Attribute group complete” received from target Principal System	Mark attribute group as complete in the system configuration.
“System Configuration Done” received from real-time system.	Pass notification of system configuration completion to the Service Layer.
“Attribute/Value pair changed” received from Real-time layer	Pass notification of attribute changes to Service Layer.

4.0 Implementation

Document [5] discusses the rational for the Systems Interface implementation decisions of document [3]. This section discusses the overall implementation of the PSBI. It is followed by a section on the Attribute/Value Layer.

Each component (task/thread/process) in the OCS that communicates with other principal systems contains the implementation of the PSBI which can include the Recipe Layer, Console Command Layer, Command Layer, and Service Layer. Using the implementation, a component can communicate with the principal systems by making calls to the Command Layer Library or the Service Layer. The OCS also uses a Service to allow one OCS application to communicate with another OCS application.

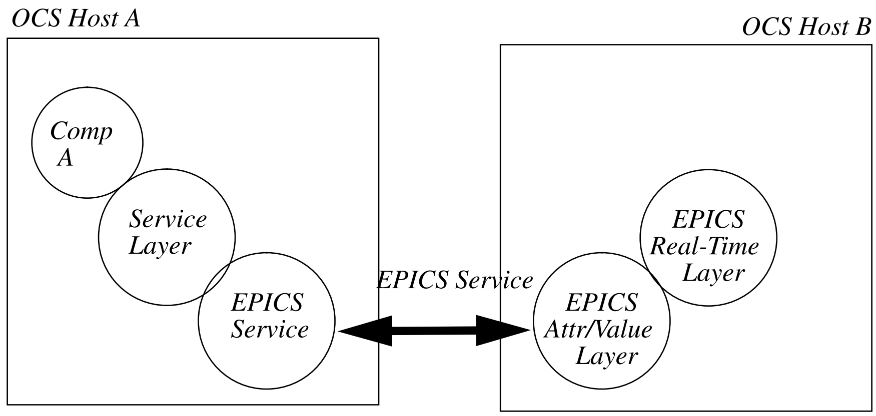
4.1 OCS Strategies

The open design of the PSBI allows for flexible composition of the OCS and the Gemini software system. The following sections show some common scenarios for connecting software system components. In the following the Command Layer is not included to make the figures more clear. However, it can be inserted between a component and the Service Layer in any of the figures.

4.1.1 OCS Communication with EPICS Principal Systems

An important software interaction occurs when the OCS communicates a command request to one of the EPICS-based Principal Systems. This case is shown in Figure 4. The OCS component passes the command to the Service Layer that uses the EPICS Service to communicate the request to the EPICS Attribute/Value layer in the target EPICS system. That EPICS Attribute/Value Layer uses the attributes to modify the activities in its Real-time Layer. The target EPICS system keeps the OCS EPICS Service up to date on the progress of the actions the command initiated.

FIGURE 4. One OCS Component communicates with an EPICS system

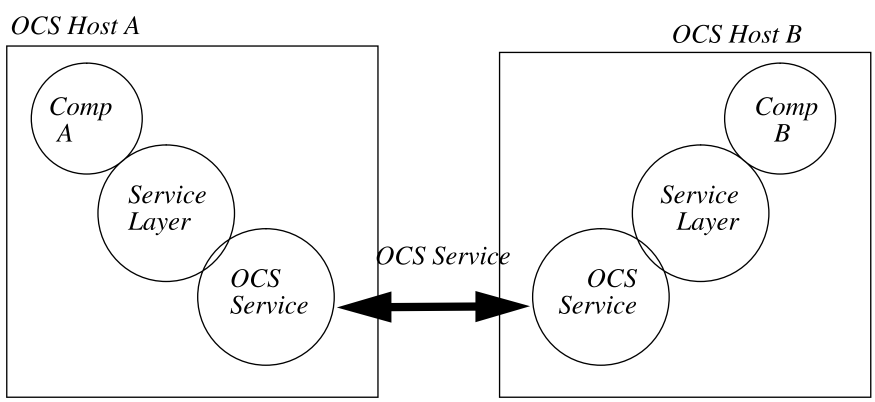


4.1.2 Two Communicating OCS Applications

Communication between OCS applications takes place using the OCS Service in the target and source. This is shown in the following figure where the sending application directly uses the Service Layer.

The request moves from the Service Layer to the OCS Service. It is received by the target OCS Service and is passed to the Service Layer and then executed in Component B.

FIGURE 5. One OCS Application sends a command to another and receives a reply.

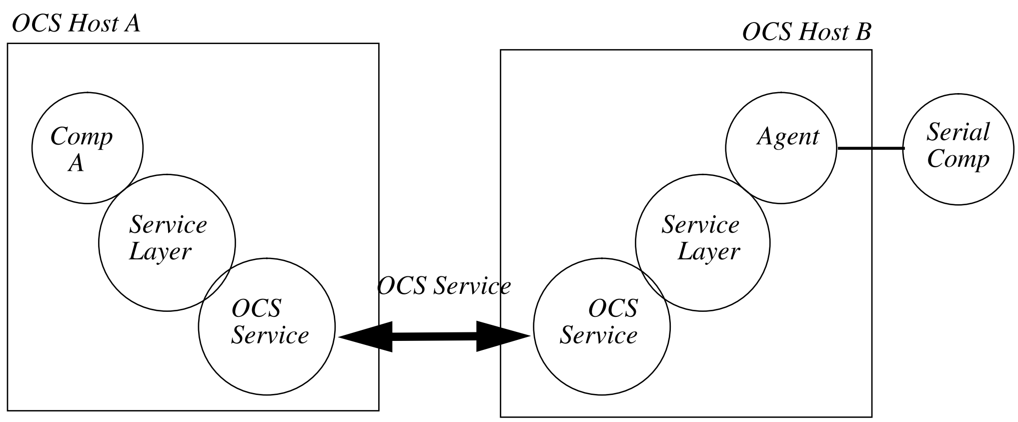


4.1.3 Use of an Agent Process

Sometimes it is useful or required to provide an application on a workstation that has the sole job of representing another software or hardware system. An application that provides this function is called an Agent. For instance, if a visitor instrument is primitive and communicates over a serial line, it can be integrated into the Gemini system using an Agent process. The Agent process is an OCS application that provides the required functionality of an OCS application and encapsulates the details of the foreign software or hardware.

This is similar to the role of a Service in the PSBI. However, as in the case of the serial device example in the previous paragraph, a Service is not always possible. A Service solution would require that applications that access the serial device run only on the host that has access to the serial line. Clearly, this is not always desirable. An agent allows any OCS application on any host to command the serial device by communicating with the device's Agent. The following figure shows how this problem is solved in the Gemini system. The Agent must provide proper Gemini behavior by hiding any incompatibilities of the serial device.

FIGURE 6. One OCS Component communicates with an Agent-based device



An Agent can be used to represent any Gemini system component including an entire Principal System if necessary.

5.0 The Baseline Attribute/Value Interface

This section describes how an Attribute/Value Layer should behave in the Gemini software system. The EPICS Attribute/Value Interface is described in detail in [2].

EPICS is a core Gemini technology and the EPICS system provides the basis for the model for command flow used in Gemini and presented in this section. It is required that all Services or Agents provide the behavior described in this section. This section is comes from [8].

5.0.1 The Action Command Model

One of the unique features of EPICS is that an IOC's activities are driven from changes to attributes in the IOC's database. The result of writing an IOC record is that *actions* occur in an IOC. An EPICs IOC has no concept of a command. Neither knowledge of the source of a command nor the command itself exists in an EPICS IOC once the command has been received and the actions started. This behavior is called the Action Command Model (ACM).

An Attribute/Value Layer and any system beneath that layer is expected to implement the Action Model.

In the Action Command Model a request message, called an *Action Directive,* is sent to a destination process and the message and its contents are immediately accepted or refused. Acceptance means the actions associated with the request will be started. Refusal of a request might occur, for instance, if a command's arguments are invalid or the related actions are already in progress and can't be altered. Once accepted, all knowledge of the sender is forgotten – the destination system is stateless with regard to commands. No information about a command is retained in a target system once a command is started. This is a fundamental of the ACM.

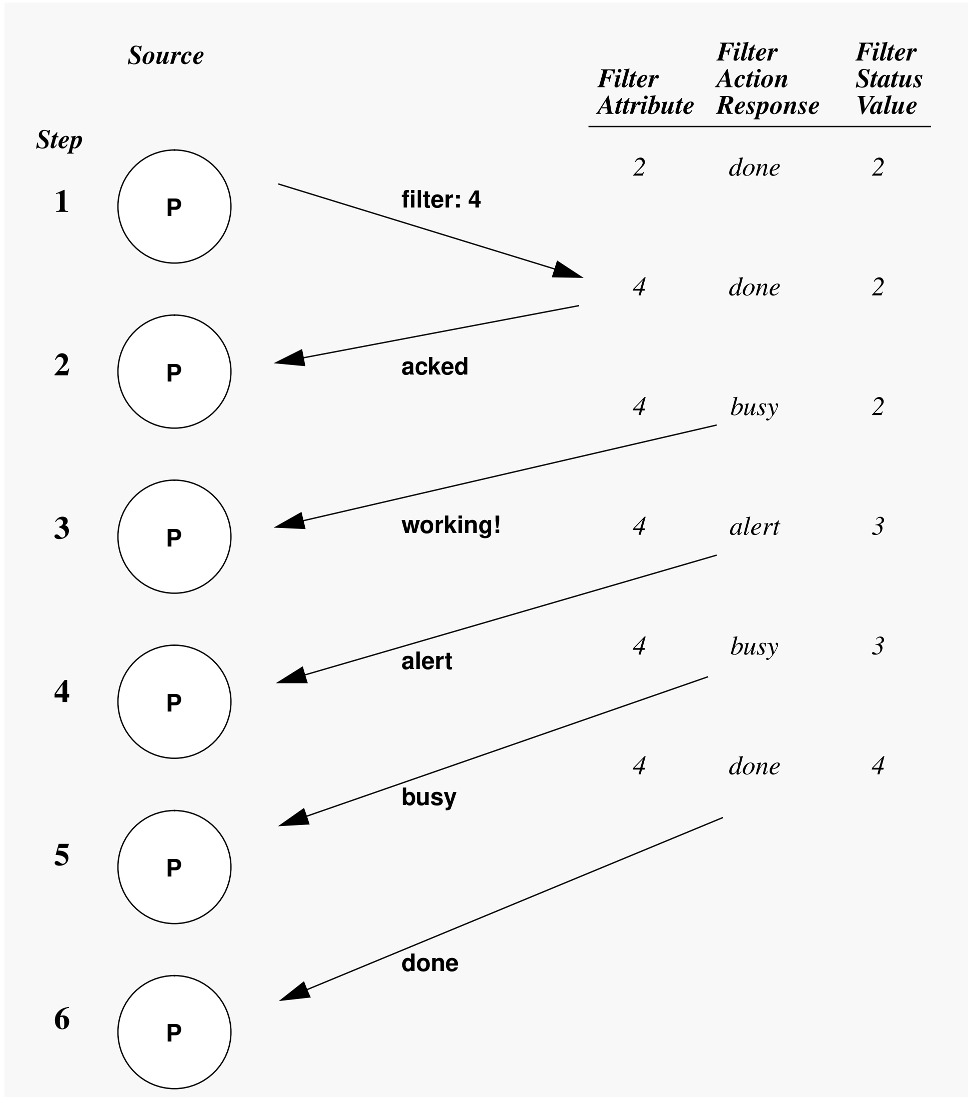
In the ACM, an *Action Response* allows the source of a request message to track the progress of the actions associated with the request. An action response is a message or database record with a value that describe the progress of one or more actions. Some possible examples for action response values are: *moving*, *busy*, *done*, or *error*. The action response values are *a type of* status value. Most status values describe the static state of a component (i.e. the current position of a filter wheel). An action response describes the activities of a particular component (hardware or software), group of components, or subsystem.

The ACM relies on defining the possible values for an action response and a protocol for their generation by a commanded system. The protocol must make sense in the variety of command situations that can occur in our control systems. An action initiated by a command may be composed of multiple 'subactions', but there must be a single action response associated with the actions. There may also be Action Responses for the subactions but these need only be monitored when efficient sequencing depends upon the completion of one or more of the subsystems. This does *not* mean that there must be a one to one correspondence between action directives and action responses.

An example is probably required. A source system is commanding the destination system to move a filter wheel from its current position (2) to a new position (4). The progress of the initial request and actions is shown in the following figure. The example is based on an EPICS system where the filter is represented by a CAD process variable that acts as the target for the action directive. A CAR record is used to provide an action response.

At Step 1, the source sends a message to the destination requesting that a filter wheel move to position 4. The command is accepted by the destination and acknowledged at Step 2. The "command" is now complete. The command acknowledgment does not come from the action response but from the commanded entity (the filter variable). It must be possible to accept and check arguments for commands independently of any ongoing actions (an example requiring this is an offset record where the argument is an incremental offset).

FIGURE 7. An Example of the Action Command Model



The destination begins the actions associated with the request once the request is accepted and the action response goes to busy and the source of the request receives notification at Step 3. Step 4 shows how an *alert* (or trigger) could be provided when the filter wheel reaches an intermediate value of 3. At Step 6, the action response monitored by the source is set to *done* - the action is complete.

The following important points are true in systems based on the Action Model.

- The command only exists in the destination software system long enough to guarantee that the actions requested in a command have been accepted.
- No state is contained in the destination that associates activities with a particular command or command source. The target should be stateless with regard to action requests.
- Source systems determine when actions are completed in a destination system by monitoring action responses that describe the progress of relevant actions.

The Action Command Model (ACM) is an approach to commands and command completion that is well attuned to EPICS.

There is no requirement that there be a one to one relationship between action directives and action responses. In the most coarse case a system might only have a single action response. In that case, a source system can only tell when *some* action is underway in the target system. As more fine-grain action responses are added, it becomes possible for the source of a command to synchronize its activities with the actions in the target system.

It is also vitally important to realize that in the ACM a specific action request can not be tied directly to a specific activity or action response. All one can say is that *some* action directive was received by the system that resulted in the actions starting. The connection between a specific action directive and an action response must be added by other software. This is done in the Gemini system through access control and client id in EPICS systems.

5.0.2 Action Model Response Protocol

The workability of the action model depends on the protocol and interpretation of values in the action responses. The initial protocol and set of values is defined here.

The protocol and value set can be represented well as a finite state machine as shown in Figure 8 on page 15. Action Response values are the states shown in capital letters. The inputs that move the entity to new action states are shown in bold. The action state of the entity is inferred from the action responses.

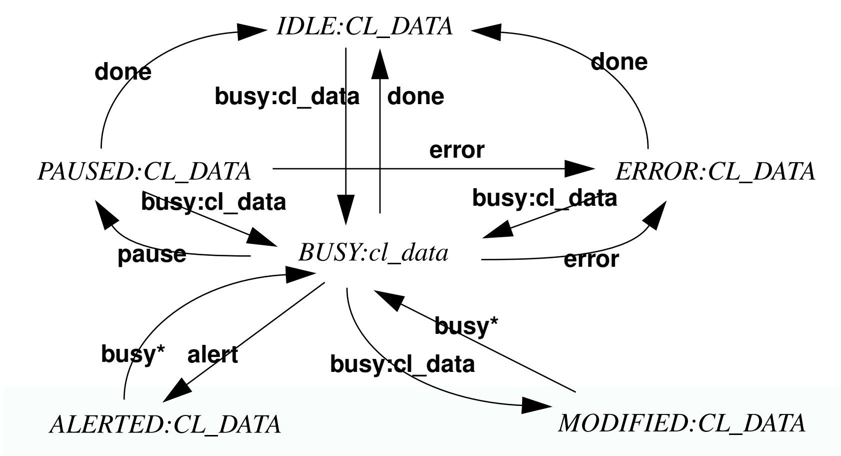
A simple action would be represented by done->busy->done with action response IDLE to BUSY and back to IDLE.

The PAUSED state is entered when an ongoing action is paused. The continue action response moves the entity to BUSY and cancel moves it to IDLE.

An ongoing action can post a message for an upper layer by issuing the alert action response and moving to the ALERTED action state and then back to BUSY.

If during an ongoing action, the entity's behavior is modified, the entity can issue the modified action response and move back to BUSY. In some cases, this might be viewed as an error, but this is determined by the entity itself, not the protocol.

FIGURE 8. Action Response Protocol



*** Indicates spontaneous decay into a state**

The ERROR action state is entered when an ongoing action fails. The failure might also make itself evident through status as an alarm. The entity leaves the ERROR action state when the action is canceled or a new action begins and the system returns to BUSY.

5.0.3 Action Model Issues

There is an important side effect of basing a control system on an action-oriented system such as EPICS that we have become aware of in discussing command models. This effect is shown in the following pseudo-TCL fragment.

```
offset 30
set waitid [offset 30]
offset 30
waitfor $waitid
do {something else}
```

The idea here is to send three 30 second offset commands, wait for the second to complete, and then do something else.

In an action model implementation, it not possible to distinguish the second offset and it is impossible to wait for just the second request to complete. (One could hack some thing to compare offset status values at the scripting layer, but we believe status values shouldn't be used to determine command completion). This situation comes up frequently. For instance, if we set a target for a slew and adjust the target while the slew is in progress, the slew action continues until the new target is obtained.

6.0 Requirements for Conforming Attribute/Value Layers

The Gemini Command Architecture is open and flexible. A Service or agent can be used to hide the details of almost any foreign software system. However, integration with the Gemini software system is easier if an Attribute/Value Layer is designed to work in the Gemini software system. This section describes minimal requirements for Attribute/Value Layers that are integral parts of the Gemini software system.

The Service Layer interface is designed to work with systems that work like EPICS. This means:

- Systems produce status values as attributes or attribute groups that are updated asynchronously. No polling should be required by the Service Layer to get values from an Attribute/Value Layer.
- In the target system, action requests are accepted, checked for feasibility, and acknowledged or rejected immediately. Once accepted, the target system is guaranteeing that the associated actions will start.

In addition, Attribute/Value Layers:

- Implement the Command Action Model.
- Produce status values and action responses.
- Use and produce control/status information that is passed as strings.
- Pass values to the Service *only* when those values change. It is the responsibility of the Attribute/Value Layer to keep track of when its values change, not the Service.

ICD1b describes the EPICS Attribute/Value layer and how it should be used in the Gemini software system.

7.0 Sequence Commands

All Principal Systems are expected to respond appropriately to the following *Sequence Commands.* More information on how these commands behave on an EPICS-based system are provided in ICD 1b.

Event (Configuration Command)	Command Arguments	Action/ Definition
test		A system should assume it has just been switched on and perform self-tests for its software and hardware systems to check that it is healthy. A **test** could be the first step in a **init** following a reboot but it is not required to be the first step. Following a **test** a system should be ready to accept commands. **test** should not require it be followed by **init** or **reset**. This command completes successfully when it passes its tests or it fails during execution and enters its ERROR state.
reboot		This command causes the system to reboot (and restart EPICS if it is an EPICS-based system). After rebooting, this system should perform the same actions as the init command (see below).
init		The system should restore itself to an initial condition, reloading initialization data. No system reboot should be performed unless that is the only way to successfully initialize the system. The command completes successfully when the system it determines it is initialized or it fails during execution and enters its ERROR state. If a system reboots as part of its **init** it must continue the **init** action following the reboot. This means that the *init* action must be set to BUSY and then IDLE following a reboot.
datum		A datum command causes a system to place its mechanisms into a state where they can be moved reliably. For mechanisms without absolute encoders, this typically means finding a reference point. The command completes successfully when the system determines it has completed reset or it fails during the process and enters its ERROR state. This command is not automatically performed on startup - it must be generated from outside the system.
park		A system should adopt an internal configuration in which it can be safely switched off. This will occur at the end of an observing session or when a principal system will enter a time of extended disuse. The command completes successfully when the system is ready to be powered down. This command fails if a problem is encountered while preparing to park. It then uses the ERROR state to return the fault.
apply		The apply command causes the system to match the configuration that has been sent to it by the OCS. More details on the behaviour of this command can be found in ICD 1b.

Event (Configuration Command)	Command Arguments	Action/ Definition
verify		This command indicates to a principal system that verification of configurations is underway by the OCS, operators, and observers. A principal system must be capable of executing changes to its state during a verify, and it must also update its status and state in the Status and Alarm Database. Interactive commands must *always* be accepted by a system. This command only provides information for principal systems and requires no special action although a system may wish to have special actions for **verify**. A principal system should successfully complete immediately after noting the **verify** command.
endVerify		This command indicates to a system that verification of configurations is finished. This command can be executed at any time. This command only provides information for principal systems and requires no special actions although a system may wish to have special actions for **endVerify.** A principal system should successfully complete immediately after noting the **endVerify** command.
guide		This command indicates to the principal systems that it should do whatever is needed to *start* the guiding operation. The command completes successfully when the **guide** actions have *begun* successfully. The *guide* action state should transition briefly to BUSY and then to IDLE once the guiding operations have begun properly and it is safe for the sequence executor to go on. Systems that choose to ignore the **guide** command should go BUSY briefly followed by the transition to IDLE.
endGuide		This command indicates to all principal systems that they should stop their guiding actions. A principal system should execute any particular behaviour that should occur when the telescope stops guiding. The *endguide* CAR action state should transition briefly to BUSY and then to IDLE. Systems that choose to ignore the **endGuide** command should go BUSY briefly followed by the transition to IDLE.

Event (Configuration Command)	Command Arguments	Action/ Definition
observe	ObservationID	This command indicates that data acquisition should begin in an instrument system based on its current internal values. Instruments executing a **observe** remain busy until they have completed the configured observation. The sequence executor uses completion of **observe** to determine when an observation completes. The OCS uses the *observe* action state to determine when the integration is complete and the data is out of the instrument and in the DHS. The *observe* action state must remain BUSY for this entire period. In addition, instruments are required to update three status values corresponding to the current phase of the observation. These are PREP (preparing to acquire), ACQ (acquiring), and RDOUT (reading out the detector and transferring data). These are detailed further in Section 7.1 on page 20. Systems other than instruments can view **observe** as informational. For those systems the *observe* action state should transition briefly to BUSY and then to IDLE. The OBSERVE command argument is an identifier that should be used by the instrument when it sends its data to the DHS. The DHS uses the ObservationID to create the data files in a predictable way (to be determined during detailed design).
endObserve		This command indicates that the configured observation has been completed. Systems may then take any steps that are needed at this point. Instrument Control Systems typically may safely ignore this command.
pause		This command indicates that a system should do whatever is appropriate for it to pause data acquisition. Pause indicates to the principal system that the user intends on continuing at a later time. The command completes successfully when the **pause** actions have begun successfully. Systems that choose to ignore the **pause** command should immediately complete successfully by setting the *pause* action state briefly to BUSY and then to IDLE
continue		This command is the reverse of pause. A system should do whatever is appropriate for it to resume data acquisition. The command completes successfully when the **continue** actions have taken place successfully. Systems that choose to ignore the **continue** command should immediately complete successfully by setting the *continue* action state briefly to BUSY and then to IDLE

Event (Configuration Command)	Command Arguments	Action/ Definition
stop		This command indicates that a system should stop the current data acquisition process normally, as if it were the end of the data acquisition period. The command completes successfully when the **stop** actions have taken place successfully. Systems that choose to ignore the **stop** command should immediately complete successfully by setting the *stop* action state briefly to BUSY and then to IDLE
abort		This command indicates that a system should stop the current data acquisition process immediately and discard any data. The command completes successfully when the **abort** actions have taken place successfully. The OCS would then notice the *abort* action state briefly go through BUSY and the *observe* action state go to IDLE. The recipe would then do whatever abort recovery is required. Systems that choose to ignore the **abort** command should acknowledge the command by setting their *abort* action state briefly to BUSY and then to IDLE.

7.1 Observing Sequence

For maximum efficiency at the telescope it must be possible to configure the principal systems efficiently. Here are some common observing situations that the OCS must be able to sequence:

- An observation consists of a series of science frames. Between frames a filter change is applied. The application of the filter change is faster than the read-out of the detector so the system is ready to start the next observe before the last read-out is complete.
- Sometimes observers will want to move to the next observation as quickly as possible, without waiting to perform detailed quality control on previous observations. In this case, the OCS can begin to configure the next observation while waiting for the detector to complete its read-out.
- Some observers do care about quality control, and for some specific instruments an apply might interfere with the detector read-out.

The common requirement in these examples is that the OCS must have more information about what is going on in the instrument in order to sequence the observations. The observe action response state alone does not offer enough granularity to handle the first two cases. It must remain BUSY the entire time that the instrument is setting up, exposing, reading out, and transferring data to the DHS.

To solve this problem, all instruments, both optical and IR, must maintain three status variables corresponding to their current activities:

- **PREP.** This status variable is ON while the detector is preparing to acquire a science frame. This is the period between the start of an observation and data acquisition, and could involve an initial reset or read of the detector.

- **ACQ.** The ACQ variable is ON while the detector is acquiring science data. For an optical detector, this is the time when the shutter is open. For IR instruments, it is the entire period while exposures are being made.
- **RDOUT.** While the detector is reading out and data is being transferred to the DHS, this variable is ON.

The OCS can use changes to these status variables to obtain finer grained control. For instance, when the ACQ flag transitions from ON to OFF, it may possible to apply a new configuration to the instrument, even if RDOUT is still ON.

Of course, this method simply makes overlapping the apply with the readout *possible*. The observer will specify restrictions on when this may occur in the observation configuration using the Observing Tool. For some instruments, it may never be a good idea to allow any activity during the readout. In this case the recipe would delay the next apply until the *endObserve* action state is IDLE.

The following sections cover a few remaining details and notes.

7.1.1 All Instruments Must Adhere to the Definitions

Much of the discussion so far has tacitly assumed an optical instrument is in use. However, for IR instruments, the definitions of PREP, ACQ, and RDOUT should remain exactly the same. ACQ should transition to ON when acquiring science data, and RDOUT should transition to ON when reading out the detector. This implies that both ACQ and RDOUT will be ON simultaneously for IR instruments, but this will not cause a problem for the OCS.

Provided every instrument follows the definitions of **observe**, PREP, ACQ, and RDOUT, the OCS will be able to sequence any instrument efficiently. Namely,

- **PREP, ACQ, and RDOUT should each transition from OFF to ON and back to OFF again *once* during an observation at the appropriate times.**

Note that this rule applies even when the instrument is paused. If the OCS cannot rely on a set pattern of transitions for PREP, ACQ, and RDOUT, then sequencing becomes much more difficult if possible at all.

7.1.2 Obtaining Header Information

With the introduction of the ACQ status variable, the OCS now has a much more accurate picture of when header information should be obtained. When ACQ goes from OFF to ON at the beginning of the observation, the OCS can snapshot header data as close to the actual time that shutter opens as possible. Likewise for the ON to OFF transition at the end of the observation.

More information on the role of PREP, ACQ, and RDOUT is provided in ICD 1b.

7.2 Sequence Command Notes

Some final notes on implementing sequence commands and future work.

Guiding Command

The **guide** command may not be adequate for the recipes that will run on the telescopes. It may be necessary to introduce additional commands or modify the definition of the **guide** sequence command once the interactive operations that take place to setup an observation are a little better known. Any changes to the sequencer commands will be under the control of the Gemini Controls Group processes and reviewed. No changes are known or expected at this time.

Ignored Commands

All systems must implement all the sequence command opcodes. The operation of the recipe in the sequence executor relies on responses from all principal systems. A principal system that doesn't care about a command should indicate completion immediately by toggling the appropriate action state to BUSY and then to IDLE.

Already There

When a system is requested to perform an action and it finds that it doesn't need to do it because the action is not required, it must still indicate completion immediately by toggling the appropriate action state to BUSY and then to IDLE. For instance, if a filter wheel is requested to move to position 4 and the controlling system notes that it is already at position four, the controlling system must still toggle the action state from BUSY to IDLE. This is true whenever any APPLY command is given.

Errors in Sequence Commands

The protocol used here defines that the only use of the ERROR state is to notify operators that an error occurred during the execution of an action. All other errors should cause the rejection of the sequence command.

Acceptance/Rejection of Sequence Commands

A principal system has the capability of accepting and rejecting the sequence commands when they arrive at the principal system. Rejection should occur if a system is unable to perform the requested sequence command. For instance, if one sent an **observe** to an instrument when it was reading out a detector, the instrument could reject the **observe**.

8.0 Debugging

8.1 Compiling Programs for Debugging

See [9] for information on compiling and linking programs that use EPICS.

See [10] for information on how to compile and link Gemini software for debugging.

8.2 Debugging Modes

A Gemini principal system or subsystem operates in any of the following debugging modes:

- NONE — There is no debugging. The system operates normally.
- MIN — There is minimal debugging. The system (perhaps) provides a commentary on some of its actions by means of the EPICS logging system but it does not carry out any operation that might severely affect its performance.
- FULL — Full debugging. The system (perhaps) provides a commentary and carries out extensive checks on its operation, which may result in a large degradation in performance.

The exact meaning of the above debugging modes is the responsibility of whoever builds the system. They may also invent further levels, as long as the three basic ones are recognized. The commentary reported by a system running in debug mode should be sufficiently explicit for a programmer familiar with the system to diagnose a problem. (For example the operator may try temporarily running a faulty system in a debug mode and report the log file to a remote programmer).

8.3 Booting and Starting

The procedures for booting and starting a VxWorks system running EPICS are described in the Standard Controller documentation, [11], and EPICS IOC developers guide, [9].

All Gemini systems should initialize themselves at debug level NONE and simulation level NONE unless specifically directed to do otherwise.

Note that EPICS has the ability to reconnect an interface automatically if one of the parties reboots.

9.0 Development and Test Factors

9.1 Acceptance Testing

Gemini systems must be able to run in a mode that allows their communication with other Gemini systems to be tested. The simulator should mimic the behavior of this interface.

9.1.1 Simulation Modes

A Gemini system or subsystem may operate in any of the following simulation modes:

- VSM — The system is to operate in 'Virtual System Mode' (a generalization of the concept of the Virtual Telescope Interface). Here, actions may be initiated and checked, but the subsystem does no further computations.

- FAST — The system's event processing and responses are enabled, and some internal computations may be performed, but response times are not realistic. Responses from lower subsystems are simulated.
- FULL — Full simulation, events and responses are enabled and system responses take realistic time. The subsystem performs all internal computations, but responses from lower subsystems are simulated with realistic timings.
- NONE — There is no simulation. The system is to operate normally.

These modes would normally be provided as each system is developed in the order shown here. The VSM mode provides a way for a client to check command flows and completeness. FAST mode allows for the testing of interfaces in an integrated environment, while FULL permits the testing of a subsystem without lower subsystems operating. The VSM mode may be omitted if a system already implements the FAST mode.

NOTE: Systems should *not* automatically fall back into a simulation mode if their hardware is not present or fails to respond. Such failures should *always* result in an error unless the system is explicitly directed into a simulation mode. This rule is necessary to ensure that simulation modes are not used accidentally, which can waste observing time.