Created: November 6, 1996
Modified: September 5, 1997 **Saved: July 27, 1998**

6

Gemini
Controls
Group
Interface
Control
Document

ICD 14 — Core Instrument Control System

Steven Beard

ICD14/07 cics_smb_035/07

This document describes the interface between a Science Instrument Control System and the CICS, i.e. it explains how an Instrument Control System can be built using the CICS as a template.

CONTENTS

1.0 Introduction

1.1 Purpose

This Interface Control Document (ICD) describes how to build an Instrument Control System using the CICS as a template.

The intended audience for this document is:

- the reviewers of the CICS;
- the developers of Gemini science instruments;
- the developers of Gemini Detector Controllers;
- the developers of any system which uses the CICS as a template (e.g. the A&G, OIWFS, HRWFS/AC or GAOS).

1.2 Scope

This document describes only how to use the template provided by the CICS. It does not lay down the law on how an Instrument Control System should be constructed. Instrument developers should consult ICD 13 [18] and Gemini Programming Standards [2] documents.

NOTE: This document is not intended to be stand-alone. It should be read in conjunction with the *"Core Instrument Control System — Introduction & Specification"*, [21], the *"CICS — Architectural Design Document"*, [22], the *"CICS — Detailed Design Document"*, [26], and the *"CICS User Guide"*.

1.3 Applicable Documents

The following documents should also be consulted:

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

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

[3] Telescopes Project.

[4] gscg.kkg.009, *"ICD 1a — The System Command Interface"*, Gemini 8m Telescopes Project.

[5] gscg.grp.024, "*ICD 1b — The Baseline Attribute/Value Interface*", Gemini 8m Telescopes Project.

[6] gscg.kkg.010, "*ICD 2 — Systems Status and Alarm Interfaces*", Gemini 8m Telescopes Project.

[7] dhs_pdr_icd3, "*ICD 3 — Bulk Data Transfer*", Norman Hill & Severin Gaudet, DAO.

[8] dhs_pdr_icd4, "*ICD 4 — Logging Interface*", Norman Hill & Severin Gaudet, DAO.

[9] gscg.grp.010, "*ICD 5 — Wavefront Sensing Information Interface*", Malcolm Stewart, Royal Observatory Edinburgh.

[10] cics_smb_017, "*ICD 6 — ICS/TCS Direct Control Interface*", Steven Beard, Royal Observatory Edinburgh.

[11] cics_smb_014, "*ICD 7a — ICS Subsystem Interfaces*", Steven Beard, Steve Wampler and Chris Mayer, Gemini 8m Telescopes Project.

[12] tcs_pbt_004, "*ICD 7b — TCS Subsystem Interfaces*", Philip Taylor, Royal Greenwich Observatory.

[13] gscg.grp.012, "*ICD 8 — Non-Conforming ICS Interfaces*", Gemini 8m Telescopes Project.

[14] gscg.grp.???, "*ICD 9 — EPICS Time Bus Driver*", Phil Taylor, Royal Greenwich Observatory.

[15] gscg.grp.020, "*ICD 10 — EPICS Synchro Bus Driver*", Andrew Johnson, Royal Greenwich Observatory.

[16] gscg.grp.021, "*ICD 11 — Event Bus*", Magnus Paterson, Royal Observatory Edinburgh.

[17] gscg.grp.017, "*ICD 12 — Gemini Interlock System*", Gemini 8m Telescopes Project.

[18] gscg.grp.???, "*ICD 13 — Standard Controller*", Bret Goodrich & Andrew Johnson, Gemini 8m Telescopes Project.

[19] gscg.grp.025, *ICD 16 — The Parameter Definition Format*, Steve Wampler, Gemini 8m Telescopes Project.

[20] ocs.kkg.031, "*Sequence Command Specifications*", Kim Gillies, Shane Walker & Steven Beard, NOAO/Gemini 8m Telescopes Project.

[21] cics_smb_002, "*Core Instrument Control System — Introduction & Specification*", Steven Beard, Royal Observatory Edinburgh.

[22] cics_smb_022, "*CICS — Architectural Design Document*", Steven Beard, Royal Observatory Edinburgh.

[23] cics_smb_033, "*ICD 1.9.3/1.9.5 — Science Instrument to Detector Controller*", Steven Beard, Royal Observatory Edinburgh.

[24] cics_smb_038, "*CICS User Guide*", Steven Beard, Royal Observatory Edinburgh.

[25] cics_smb_039, "*CICS Project Glossary and References*", Steven Beard, Royal Observatory Edinburgh.

[26] cics_smb_041, "*CICS — Detailed Design Document*", Steven Beard, Royal Observatory Edinburgh.

[27] ag_jms_001, "*A&G Control System vis-a-vis Other Gemini Systems*", J.M.Stewart, ROE.

[28] ag_jms_004, "*Control of Wavefront Sensors*", J.M.Stewart, ROE.

[29] niri_hty_001, *“Near IR Imager (NIRI) - Architectural Design Document”*, Hubert Yamada, Institute for Astronomy, UNiversity of Hawaii, Honolulu.

[30] niri_hty_002, *“Near IR Imager (NIRI) - Detailed Design Document”*, Hubert Yamada, Institute for Astronomy, UNiversity of Hawaii, Honolulu.

[31] *WHT ADAM/EPICS INTERFACE*, Version 0.2, Paul Martin, Chris Mayer, Phillip Taylor, 27 July, 1994. Available from the authors.

[32] gscg.bdg.007.dsup, *“Writing EPICS Device Support”*, Bret Goodrich, Gemini 8m Telescopes Project.

[33] EPICS *Display Manager User’s Guide*, Los Alamos National Laboratory.

[34] EPICS *Archiving Reference Manual* (DRAFT), Roger Cole, Los Alamos National Laboratory. [N.B. This document is in preparation. The Gemini project has a draft copy. Contact the author for up to date information].

[35] EPICS *Channel Access Reference Manual*, Jeffrey O. Hill, EPICS Group, Los Alamos National Laboratory.

[36] EPICS *Input/Output Controller Record Reference Manual,* J.B. Anderson, M.R. Kraimer, October, 1992.

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

[38] EPICS *State Notation Language and Run-time Sequencer Users Guide,* Andy Kozubal, Los Alamos National Laboratory

[39] CAPFAST *Electronic Circuit Design CAE User’s Manual,* Phase Three Logic Inc.., Beaverton, Oregon.

1.4 Abbreviations and Acronyms

A&G	Acquisition and Guidance
AC	Acquisition Camera
AO	Adaptive Optics
CA	Channel Access
CAD	Command Action Directive
CAR	Command Action Response
CC	Components Controller
CICS	Core Instrument Control System
CRS	Cassegrain Rotator control System
DC	Detector Controller
DHS	Data Handling System
ECS	Enclosure Control System
EPICS	Experimental Physics and Industrial Control System
FITS	Flexible Image Transport System
HRWFS	High Resolution Wavefront Sensor
GAOS	Gemini Adaptive Optics Subsystem
GBD	Guidance and Beam Direction (part of the A&G)
GCS	Gemini Control System
ICD	Interface Control Document
ICS	Instrument Control System
IOC	Input Output Controller
IS	Instrument Sequencer
LAN	Local Area Network
M1	Primary Mirror
M2	Secondary Mirror

MCS	Mount Control System
OCS	Observatory Control System
OIWFS	On Instrument Wavefront Sensor
PCS	Primary mirror (M1) Control System
PDF	Parameter Definition Format
SAD	Status/Alarm Database
SCS	Secondary mirror (M2) Control System
SDD	Software Design Description.
SIR	Status/Information Record (EPICS)
TBD	To Be Decided
TCP/IP	Transmission Control Protocol/Internet Protocol
TCS	Telescope Control System
UDP/IP	User Datagram Protocol/Internet Protocol
VSM	Virtual System Mode
WAN	Wide Area Network
WFS	Wavefront Sensor

1.5 Definitions

VME	A real-time system obeying the ANSI/IEEE 1014-1987 Versatile Backplane Bus standard.
VxWorks	The Real Time Operating system from Wind River

1.6 Stylistic Conventions

References to documents in Section 1.3, “Applicable Documents,” on page 2 are given like this; [1].

Note that, unless otherwise specified, the term “Detector Controller” refers to the Detector Controller *software system*.

2.0 Overview

An overview of the Gemini Control System is given in the Gemini *Software Design Description*, [1], and an overview of the structure of an Instrument Control System is given in the introduction to the Core Instrument Control System (CICS), [21] and the CICS Architectural Design, [22]. The detailed design of the CICS is described in the CICS Detailed Design [26]. An overview of the A&G and wavefront sensing systems is given in [27] and [28].

2.1 System Hardware Architecture

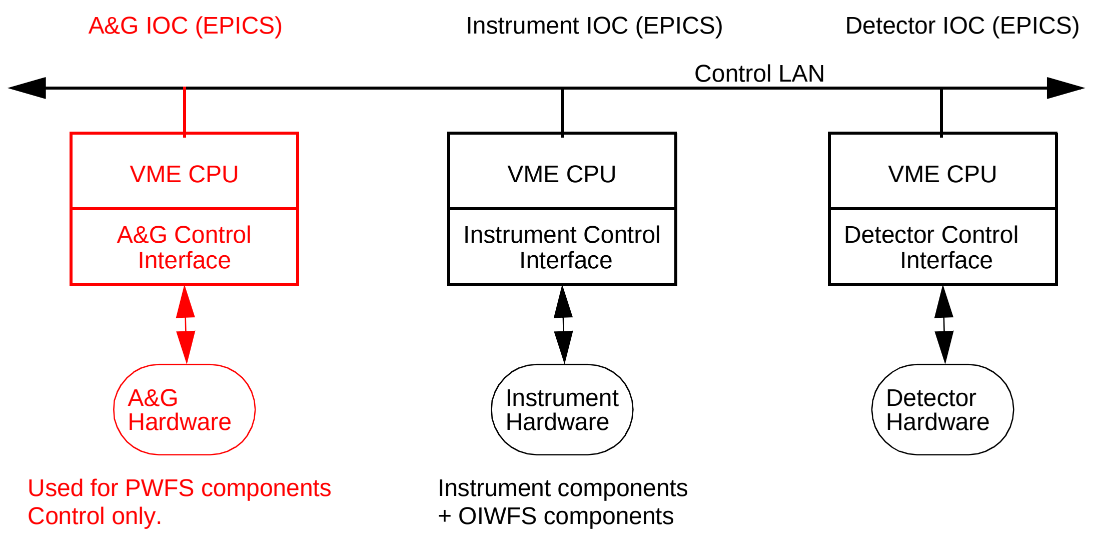
An Instrument Control System uses a VxWorks/VME based IOC using EPICS for internal communications and control. Each ICS consists of one or more VxWorks systems supporting the Instrument Sequencer (IS) software, the Detector Controller (DC), and the Components Controller (CC). A full description of the Gemini hardware architecture is contained in ICD 13, [18].

In practice, the Instrument Sequencer and Components Controller will all run on the same EPICS Input Output Controller (IOC), and the Detector Controller will be in a separate EPICS IOC, as shown in Figure 1 on page 6. If the Gemini project chooses to use an existing Detector Controller, rather than building one in-house, this might not have the same architecture as shown in Figure 1. It will at least have to act like an EPICS channel access server at the interface with the Control LAN in order to conform to this ICD.

If an instrument has an on-instrument wavefront sensor (OIWFS), the Components Controller for this OIWFS will be installed on the instrument's IOC but will *not* be controlled by the Instrument Sequencer. Instead control will come from the A&G GBD sequencer on a separate IOC, as shown in red on Figure 1. (Note that the Detector Controller for an OIWFS is installed entirely on the A&G IOC). See [27], [28] and [9] for details.

The ICS 'Instrument Sequencer' software is responsible for managing the interfaces between all the instrument systems and other Principal Systems (OCS, DHS and TCS). These interfaces are not covered by this ICD.

FIGURE 1. The Instrument Subsystem Command, Status and Alarm Hardware Architecture



2.2 Communication Architecture

The ICS systems communicate internally using EPICS Channel Access across a high performance LAN (the Control LAN). Communications are established using UDP, but run using TCP/IP.

2.2.1 ICS Context Diagram

See the CICS Architectural Design Document, [22].

2.2.2 ICS Events and Responses

See the CICS Architectural Design Document, [22].

2.3 The Services Component of the Interface

2.3.1 Communication Services and Protocols

All communication takes place using the EPICS Database Access or Channel Access Protocol [37]. Communication between separate EPICS boxes takes place by means of TCP/IP messages communicated on the Local Area Network.

2.3.2 Host Support Services

N/A

2.3.3 Target Support Services

N/A

3.0 Implementation

The CICS has been designed to be a template (see [22] and [26]). It simulates the control of hardware and has an arbitrary selection of components. The CICS needs to be adapted for the requirements of each particular instrument, as described here.

To help with the integration of the CICS and a real instrument, comments preceded by the string “ICD 14” have been included in the CICS source code, state notation language and Capfast diagrams. Consulting the CICS Detailed Design, [26], and searching for the string “ICD 14” should help identify the changes necessary for a particular instrument.

3.1 Use of the CICS in a Components Controller

The following steps are required to make use of the CICS in the Components Controller an actual Instrument Control System:

1. The top level CICS Capfast schematics need to be modified to reflect the actual components making up the instrument (rather than the arbitrary selection chosen for the CICS).
2. The top level CICS Capfast schematics also need to be modified to reflect the actual class of each type of mechanism. For example, if the focus mechanism operates in discrete steps rather than continuously, then the focus needs to be represented by the “one axis discrete mechanism” (comp1d) symbol rather than the “one axis continuous mechanism” (comp1c) symbol.

   If the mechanism is of a class which does not yet exist in the CICS, then a new schematic for that class needs to be developed (e.g. a three axis mechanism with two continuous axes and one discrete axis could be described with a “comp3ccd” schematic). If new classes of mechanism are developed these should be made available to other instrument groups and provided to IGPO for incorporation into the CICS collection.
3. The CICS simulates motor control in its “motor.sch” schematic. This simulates a motor by attempting to make the “actual” position stored in one EPICS record the same as the “demand” position stored in another record. In a real instrument this schematic needs to be replaced by an actual motor control schematic. If all motors are controlled in the same way it might be sufficient just to replace this one schematic with a motor control database. If the instrument uses different kinds of motor then it may be necessary to provide a separate motor control database for each class of motor. A motor control database is likely to contain the following:
   – A stepper motor record (or a record designed for that particular kind of motor).

- Records containing the actual and demand positions for the motor (if these are not already available as part of the stepper motor control record).
- Records, such as the “Sequence”, “Scan” and “Wait” record, responsible for sequencing the motor when carrying out high level operations such as a datum search.
- Binary records connected to microswitches which detect the passage of particular reference points.
- If the mechanism has absolute encoders there may be records connected to those to report the actual location of the motor.

In general, simple sequencing operations (such as moving the motor until a microswitch is triggered) can be carried out in the motor control database. Complex operations involving the coordination of more than one motor may be better done in the EPICS state notation language (see next item).

Instrument groups are free to use the CICS mechanism simulator for simulating their mechanisms while in “FULL” simulation mode.

4. The CICS requires any motor control database to have the following inputs and outputs:

   Inputs

   - Demand position in engineering units;
   - Start/Stop operations;
   - “Find datum” operation;
   - Maximum velocity (optional).

   Outputs

   - Busy flag;
   - Current position in engineering units;
   - Current velocity (optional);

   The CICS can get away without using the optional parameters. Some instruments may need to return additional parameters (such as a movement status and message), and the CICS can be adapted to handle these. See the NIRI case in SECTION XXXX.

5. The state notation language which controls the mechanisms may need to be altered to reflect the actual inputs and outputs of the real mechanism. If the mechanism recognises additional commands (in addition to “move”, “init” and “datum”) those commands need to be added to the state notation language. See the NIRI case in SECTION XXXX for an example.
6. The state notation language associated with a particular class of component may also need to be altered to reflect the actual properties of that component. For example, if the CICS assumes the mechanism needs to be datumed but the mechanism actually has absolute encoders, then the “not_datumed” and “searching_for_datum” states can be removed and the mechanism can begin in the “idle” state once initialised.

   Classes of mechanism which require complex sequencing (for example the control of a two axis probe in which there are some areas of avoidance in two dimensional space) would need to be controlled by more complex state notation code than provided with the CICS. The CICS only provides a basic control system, and it is each instrument team’s responsibility to develop the extra code needed to meet their own instrument’s requirements. New classes of state notation code should be shared with other instrument groups who have similar problems to solve.

3.1.1 Scaling the CICS Components Controller

Often it may be necessary to adapt the CICS Components Controller to deal with a completely class of mechanism, for example a three axis continuous mechanism. The existing mechanism classes can be scaled up to make new classes. For example, the two axis continuous mechanism may be converted to a three axis continuous mechanism by the following procedure:

1. Make a duplicate copy of the following "comp2c..." schematics, renaming them as follows and changing all the comments so they refer to a three axis mechanism:

Old name	New name
comp2c.sch	comp3c.sch
comp2ccCad.sch[a]	comp3cccCad.sch
comp2Car.sch	comp3Car.sch
comp2Mech.sch	comp3Mech.sch
comp2cSad.sch	comp3cSad.sch

a. Note. The "cc" is not a typing mistake. It refers to the fact that the mechanism is continuous along both its axes.

2. Edit all the new schematics with a text editor (quicker and easier than using Capfast) and ensure all the above schematics they refer to are renamed as above.
3. Edit "comp3cccCad.sch" and add another instance of the "compcMoveAxis.sch" schematic with variables $(axis3) and $(units3), then wire this new schematic to the Move and Park CAD records.

   Rewiring this schematic may require several "mfanout" schematics to be built into a hierarchy, or a new version of the "mfanout" schematic to be produced.[1]
4. Edit "comp3Car.sch" and add a new instance of the "compCarAxis.sch" schematic with variable $(axis3). Wire this new schematic to the input of the "genSub" record.
5. Edit "comp3Mech.sch" and add a third motor with variable "motor" set to "3".
6. Edit "comp3cSad.sch" and add a third instance of the "compcSadMotor.sch" schematic with variables $(axis3) and $(units3) and wire this new schematic to the inputs of the "calc" and "genSub" records. Then edit the CALC field of the "calc" records so the third input is included in the logical test, viz:

   ```
   A=0?{B=0?{C=0?0:1}:1}:1
   ```
7. Add a new mechanism to the under top level Components Controller schematic using "comp3c.sch" and set up variables something like this:

   ```
   set1 mechanism translation/focus
   set2 mech dxyz
   set3 units1 millimetres
   set4 units2 millimetres
   set5 units3 millimetres
   set6 axis1 X
   set7 axis2 Y
   set8 axis3 Z
   set9 hindex $(sadtop)combHlt.I
   set10 mindex $(sadtop)combHlt.J
   ```

1. It is unfortunate that, while I have designed the CICS to be as expandible as possible, this method of producing software by wiring up Capfast diagrams makes it hard to expand existing software.

```
set11 cindex $(sadtop)combCar.I
set12 hindex $(sadtop)combCar.J
```

8. Finally, add the same mechanism to the under top level Components Controller Status Alarm Database schematic using “comp3cSad.sch” and set up the same variables as above.

3.2 Use of the CICS in a Detector Controller

The Detector Controller parts of the CICS are not as well-developed as the Components Controller parts, and there are more variations possible. The basic rule is that a Detector Controller based on the CICS must obey the Gemini ICDs (e.g. ICD 1.9.3/1.9.5, [23]). The Detector Controller must be commandable through CAD records, and it must be possible to monitor its status through CAR and SIR records.

Some Detector Controller operations, such as the one-off sensing and setting of voltages and the sensing and throwing of switches, map well onto the facilities provided by EPICS. The CICS contains an example of the setting of a bias voltage and a clock voltage. However, there are other operations in a Detector Controller which do not map well onto the EPICS facilities, and at the bottom level of a Detector Controller there is likely to be a lot of pure C/VxWorks code. The following options are possible for interfacing this C code to the EPICS system provided with the CICS.

- Take the CAD records defined by the CICS, add some more according to the requirements of each particular Detector Controller, and call the C/VxWorks code belonging to the Detector Controller directly from the subroutine executed in the CAD records.
- Write some EPICS device support for the Detector Controller, as described in [32], and activate that device support through EPICS records.
- Add some “genSub” records to the CICS Detector Controller and call the C/VxWorks code from those.
- Enhance the state notation code provided with the CICS and call the C/VxWorks code from within that. This option is useful if the Detector Controller has to monitor flags or respond to outside events.

There is no reason why a Detector Controller cannot make use of all three of these modes.

The data handling side of a Detector Controller may be a lot more complex than provided by the CICS. The information and example code provided in ICD 3, [7], can be followed.

4.0 The Programmatic Interface

Instrument groups can make use of the utility libraries, utility Capfast schematics and state notation code provided with the CICS. These are described in the “CICS Detailed Design Document”, [26].

5.0 Debugging

Same as ICD/7a, [11].

6.0 Error, Alarm and Logging System

Same as ICD/7a, [11].

7.0 System Attributes

Same as ICD/7a, [11].

8.0 Development and Test Factors

Same as ICD/7a, [11].

9.0 APPENDIX — NIRI Case History

The Gemini Near Infrared Imager (NIRI), described in references [29] and [30], is the first instrument to use the CICS as a template. It was surprisingly easy to interface the CICS and NIRI together because the NIRI mechanism control system closely matched the interface described in Section 3.1 on page 7. The NIRI mechanism control database contained a special purpose EPICS record which handled the NIRI motors and hall effect sensors. The actual NIRI control software is written using C-code as EPICS device support, [32]. Each NIRI mechanism used the following interface with the CICS:

Inputs

- Demand position in engineering units (which issues an automatic “move” operation);
- “Stop” operation;
- “Find datum” operation;
- “Diagnose” operation.

Outputs

- Operation BUSY or IDLE flag;
- Operation completion status (OK or error);
- Operation completion message;
- Mechanism datumed flag;
- Current position in engineering units;
- Motor MOVING or STOPPED flag.

The following changes were necessary to the CICS to adapt it to NIRI:

1. The CICS mechanism simulators were replaced by the NIRI mechanism control database, one instance per NIRI mechanism.
2. The CICS state notation language (SNL) was modified to adapt to the NIRI interface and to NIRI requirements. In particular:
   - The SNL was adapted to use the NIRI input and output records for demand position and current position, etc...
   - A “stopping” state was added to cope with the fact that real mechanisms take a finite time to stop once a stop command has been received.

   - Whenever a mechanism stopped or an operation completed the NIRI status flags were read to determine whether the operation was successful.
   - The SNL was modified to detect changes in the NIRI “mechanism datumed” flag, and the state switched between “datumed” and “not_datumed” accordingly.
3. The CICS C code was also modified as follows:
   - For NIRI it is not acceptable to datum a mechanism automatically if a move command is received before the mechanism is datumed. The CAD functions for moving the mechanisms were therefore modified to accept a “datumed” flag on one of their inputs and reject the command if the mechanism has not been datumed.
4. The top level Capfast schematics were modified to reflect the actual names and classes of the NIRI mechanisms.
5. The PvLoad parameters files and lookup table files were modified to reflect the actual allowed values and ranges of the NIRI mechanisms.

Interfacing NIRI and the CICS together roughly took about one week, and tidying up all the loose ends took another 3 weeks. So the total effort required to interface the CICS and NIRI together was about one month.

The NIRI source code can be consulted as an example of how to adapt the CICS for a real instrument.