UK Astronomy Technology Centre
Royal Observatory, Edinburgh,
Blackford Hill, Edinburgh, EH9 3HJ, UK.

Andy Vick (ajv@roe.ac.uk)
Last save 01/11/99 1:52 PM
AOW/SOF/ajv/1.1/01/99 **wht-naomi-38**

Request for comment
Interface Control Document: NAOMI to Drama Translator

Notes

This document is only concerned with high level software interfaces between the sub-systems listed, other interfaces (electronic or sub-system internal) are not covered.
The intention of this document is to solicit comments from all parties involved in the design or development of NAOMI about the interface requirements between the following sub-systems (The names in brackets are the names by which these sub-systems are identified in this document);

The high level control tasks that will control the actions of NAOMI.	(NAOMI/OBSERV)
The sequencer sub-system that controls the AO loop. Note – I have labeled all Durham software within the real-time system as SEQ, whether or not it is a part of the Sequencer.	(SEQ)
The EPICS sub-system that controls the opto-mechanical chassis package.	(EPICS/OMC)
The EPICS sub-system that controls the wavefront sensor package	(EPICS/WFS)
The Drama sub-system that controls the INGRID science instrument	(Drama/INGRID)
The TCS control link.	(TCS)

Whilst this document is not complete as an ICD, it already contains a lot of assumptions and the possibility of a lot of misconceptions, so any comments are useful!

1 Envisaged use

1.1 NAOMI/OBSERV

The NAOMI/OBSERV sub-system will be used for all top-level control within the NAOMI system, and as such will be a sub-system within the Observatory control system. All commands to the NAOMI system, which are required for normal (non-engineering) use of the system, will be sent via Drama. It is expected that most of the sub-systems of NAOMI will communicate with one another via Drama in some manner.

1.2 SEQ

The sequencer part of NAOMI will be used to control the internals of the real-time sub-system, and the engineering and diagnostic functions of various parts of the system. As such it provides a simple interface to the functions of the sub-system, so that observatory level tasks can issue a set of high level commands and do not have to worry about the minutiae of controlling the NAOMI system. High level commands to the SEQ will come via Drama, internally the SEQ uses a combination of Python, DTM and a custom messaging format (GP) to transfer messages.
I have assumed that it will not be necessary for the sequencer to send GP messages outside of the real time part of the system, so the only messages that the NAOMI to Drama translator will need to handle (on the sequencer side) are Python/DTM.

1.3 EPICS/OMC

There is an early version of the EPICS/OMC ICD[1], but this document will be superceded by a new ICD, written for the existing EPICS/OMC, which should be available in March '99. In the meantime, I have

assumed that the EPICS/OMC interface is a set of EPICS records accessible from Drama via the Edif library.
I have also assumed that all control can be achieved with (1 dimensional) array variables and that no callbacks to the SEQ sub-system will need to be performed.

1.4 EPICS/WFS

The EPICS/WFS sub-system is similar to the EPICS/OMC system in its design and usage, being a single Drama task that interfaces via Edif to EPICS (using a modified version of SDSULib). The software interface does not need to handle any image data for NAOMI itself, although there may exist a requirement to access data directly from the WFS for testing work at ROE.
It is assumed here that all control can be achieved with array variables and that no callbacks to the SEQ sub-system will need to be performed.

1.5 Drama/INGRID[9]

Drama/INGRID is not a sub-system of NAOMI, being a science instrument in its own right, so the interface has been designed for incorporation in the Observatory's control and data acquisition systems. In addition to this, the sub-system also needs to be able to provide image data to the SEQ sub-system in a rapid readout mode[1] (less than 0.1 seconds for a 128x128-pixel image). Aside from this criteria it is again assumed that all control can be achieved with array variables and that no callbacks to the SEQ sub-system will need to be performed.

1.6 TCS[8]

Whilst in operation, NAOMI assumes the task of updating position and focus information to the TCS, and therefore needs to set, read and check the validity of these data. Position and focus control also has to be available to INGRID during normal operation, requiring some sort of lock managing between the two systems. It is assumed that the high-level control system (NAOMI/OBSERV) will handle these issues, and that a simple Drama interface task can be supplied for communications to and from the TCS.

2 System communications layers

The following table shows the proposed communication layers that various sub-systems will use in order to transfer data.

SEQ	**EPICS/OMC**	**EPICS/WFS**	**Drama/INGRID**	**TCS**	
Drama Python-DTM	Drama EPICS	Drama EPICS	Drama	Drama	**NAOMI/OBSERV**
	Drama EPICS Python-DTM	Drama EPICS Python-DTM	Drama Python-DTM	Drama Python-DTM	**SEQ**
		NA	NA	NA	**EPICS/OMC**
			NA	NA	**EPICS/WFS**
				Drama	**Drama/INGRID**

2.1 Drama/EPICS.

The EPICS Drama Interface[2] (EDIF) is a single Drama task that mirrors a set of EPICS records to Drama parameters. At startup, the EDIF task reads a text file to determine which EPICS records it should install

EPICS callbacks on (EPICS records can also be added to EDIF using a reload command). When the values of these records change, the callback is used to change the value in the Drama parameter. Changes to the Drama parameter from other Drama tasks are sent to the EPICS record (using ca_put( )).
Note - Because the Drama and EPICS interfaces are buffered, both Drama and EPICS update routines need to be called at a pre-determined rate, which means that there is a certain dead-time between the updates occurring.
There are still some outstanding questions related to the Edif interface which may influence the NAOMI to Drama translator, which are listed here until they can be resolved;

1. Does the Edif interface map array's, or only scalar values.
2. Does Edif return the EPICS return codes (e.g. ECA_NORMAL) or are these mapped to Drama return codes in some form.

2.2 Python/DTM

The Python/DTM layer provides library calls to send and receive DTM messages using the Python interpreter to parse the message header into command names and arguments. The DTM library provides a socket based messaging layer that provides ASCII message headers with network independent blocks of data. The message header used is a Python method (function) call[3];

```
  class.method( argname1=value1, argname2=value2, ..., argnameN=valueN)
```

C code library routines are provided to encode and decode messages.
The values in the method call can either be ASCII strings or DTM data blocks; the access routines can be used to extract the data either way.
The library provides two basic methods for accessing messages;

?? A simple send and receive message service, where the message receive has to state what name (class & method) that it wishes to get.
?? An application loop interface, where callbacks are defined for different message types.

In both cases the returned message has its arguments converted to a Python dictionary object, access routines are provided for looking up arguments in the dictionary[4].

3 Interface Specifications

The following are based on the concept of a single interface mechanism, running as a server, with two communication interfaces, one which connects to Drama services and one DTM/Python.

3.1 General.

1. The interface software should run as a server.
2. The interface software should allow status checks to be performed remotely from either Python/DTM or Drama services.
3. The interface should provide status returns to appropriate services in the case of any communications or system failures.
4. The interface latencies must be small enough to provide the rapid readout mode of INGRID (less than 0.1 seconds for a 128x128-pixel image).

3.2 Drama side interface.

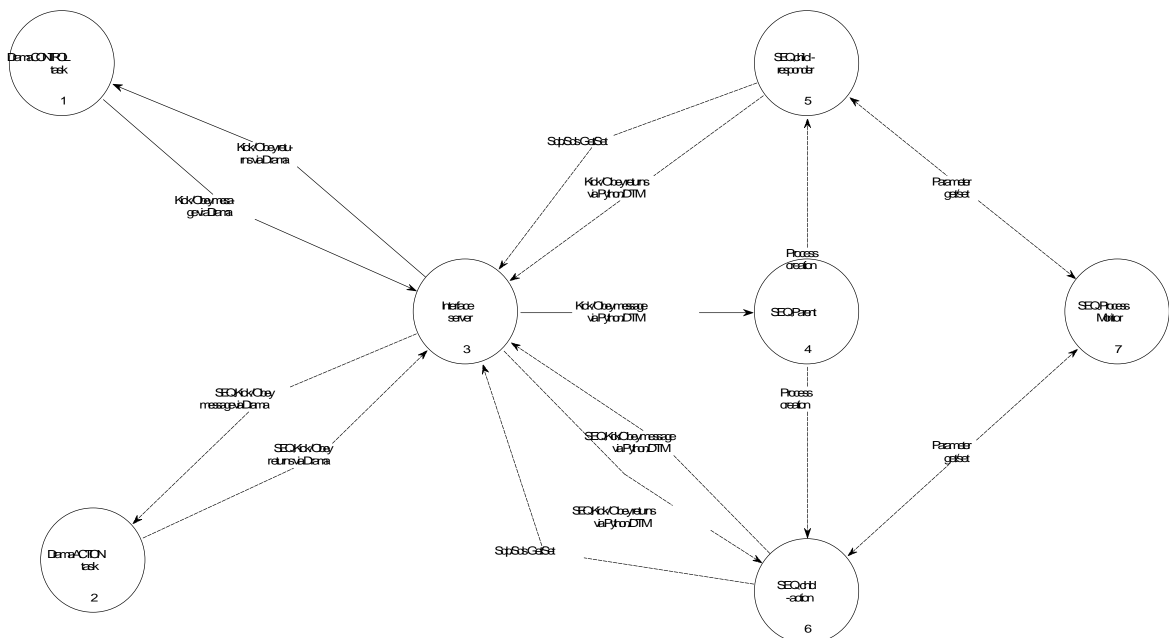
1. The Drama side of the interface will appear as a single Drama task, with actions that are dynamically allocated by the SEQ.
2. The Drama side of the interface needs to be able to convert Drama Kick and Obey messages, including parameters, into the Python/DTM format and forward them to the SEQ.
3. Messages returned as the result of Kick or Obey messages need to be returned to the task that initiated the message.

3.3 Python/DTM side interface.

1. SEQ must be able to “announce” which actions it wishes to respond to[6].
2. The interface needs to be able to convert Python/DTM messages into Drama Kick or Obey messages and parameters, and forward them to Drama tasks.
3. Messages returned as the result of Kick or Obey messages need to be returned to a specific Drama task, which will be waiting (rescheduled) at the interface.
4. The Python side needs to be able to access Drama parameters, both to set and get values.
5. The Python side needs to be able to access Sds Image Structures (see DitsGetArgument()).
6. The Python side needs to be able to send Kick and Obey messages to other Drama tasks, and to receive any messages returned (Transaction completion, Transaction failure, Trigger, Informational and Error).

4 Interface description.

The following diagram indicates the communication channels that will operate via the interface to allow Python/DTM and Drama to action each other’s tasks and to exchange parameters;



The following is a list of all the message headers that the interface and Python-DTM will need to pass to one another in order to implement the access to Drama that is required. Note that these strings, although they are formatted as valid Python statements, are not the calls themselves, just headers that describe the DTM messages and what the message body contains.

Notes:

1. Only simple arguments (strings, ints, floats etc) will be converted for the Obey and Kick messages, C structures will be (silently?) ignored.
2. The item after the message description indicates where the message originates from, and where it’s intended destination is.
3. It might be better to differentiate between the directions of messages in the name, e.g. DramaMsgIn.whatever for Drama to SEQ and DramaMsgOut.whatever for SEQ to Drama.
4. The naming convention for the portname argument (which indicates the port that DTM will use to access a returning message) isn’t specified.

The Python side of the interface will accept the following messages in Python/DTM format;

DramaMsg.GetStatus(status=value, portname=value, currenttasks=listvalue) SEQ to Interface

Gets the status of the interface, and a list of current tasks awaiting completion.

DramaMsg.AddAction(task=name, portname=value) SEQ to Interface

Adds a new action to the interface, intended to be used an initialisation only.

DramaMsg.GetParam(task=name, portname=value, param=name) SEQ to Drama

Accesses the named parameter from the named task and returns its value. Only simple parameters (strings, ints, floats etc) and images will be accessed, C structures etc will cause an error return.

DramaMsg.SetParam(task=name, portname=value, param=name, value=datum) SEQ to Drama

Accesses the named parameter from the named task and sets its value. Only simple parameters (strings, ints, floats etc) and images will be accessed, C structures etc will cause an error return.

DramaMsg.Obey(task=name, portname=value [, argument1=value, ..]) SEQ to Drama

Actions a call to DitsObey() on the named (Python) task. Arguments are converted into a new Drama argument structure and sent in the DitsObey() call.

DramaMsg.Kick(task=name, portname=value) SEQ to Drama

Actions a call to DitsKick() on the named (Python) task.

DramaMsg.Return(reason=value, portname=value, status=value). Drama to SEQ

Return from the Drama task that was sent an Obey or Kick message.

Note – since the Drama path lookup is done by the interface, an extra reason value of "No such path" has to be returned by the server.

The Drama side of the interface will map Obey and Kick calls to one function, which will action the routine specified in the lookup table (as set by DramaMsg.AddAction). Actioning the routine will consist of sending one of the following Python/DTM messages to the sequencer;

DramaMsg.Kick(task=name, portname=value) Drama to SEQ

DramaMsg.Obey(task=name, portname=value [, argument1=value, ..]) Drama to SEQ

Messages sent to the SEQ as a result of an incoming Obey or Kick message from Drama.

All the arguments obtained from the Obey or Kick message are converted to arguments in the Python call. After one of these messages has been sent, the routine will post the call to a lookup table, and await a Python/DTM return message) to forward back to Drama. Using dynamically allocated Drama actions in this way will allow the interface to easily keep track of what has been sent to the SEQ (since Drama serializes messages to the same action). It also means that the interface will not need to be recompiled as features of the system are added or changed.

DramaMsg.Return(reason=value, portname=value, status=value). SEQ to Drama

Message sent back from the SEQ as a reply to the Drama Kick or Obey message.

Appendix 1

A quick introduction to the Edif interface (written by a novice).

At initialisation the Edif main loop performs the following

i. Creates a new Sds structure named "PARA" which will hold copies of EPICS records for access by other Drama tasks
ii. Adds action handlers for the following;

Action name	Function in edif.c
INIT	Setup()
RELD	ReLoad()
SET	Put()
FLUSH	Timer()
EXIT	Exit()

iii. Initialises Dits and Sdp
iv. Runs **DitsMainLoop()**
v. Ends by returning the result of calling **DitsStop()**

Starting and accessing the Edif Interface.

1) Find the path to the Drama task.

The Drama task for the interface is called "EDIF" (set via the macro TASKNAME in edif.c), so to initialises talking to the interface via Drama call **DitsGetPath("EDIF",...)**.

2) Tell the interface to initalise its access to EPICS.

This must be done before you start trying to access any EPICS records. To do this you must action "INIT", with an (optional) filename argument "Argument1";

```
DitsArgType id;
ArgNew(&id,status);
ArgPutString(id,"Argument1","/home/username/my_setup_file.txt",&transic
              status);
DitsObey(edifPath,"INIT",id,&transid,status)
```

If you omit the argument, the interface uses a default filename "records.names", and prints a message to that effect to stdout. If the interface determines that it already has records (i.e. "INIT" has already been actioned) then it carries on with the initialisation, but warns you with a MsgOut().

3) Setup the interval between calls to ca_pend_io() and SdpUpdate().

Both Drama and EPICS buffer access to their data items, so these two functions have to be called periodically. In order to initialise this, the action "FLUSH" must be called once to start this happening, i.e.
DitsObey(edifPath,"FLUSH",id,&transid,status)
In version 1.0 of EDIF the timer frequency is set via the macro MICRORATE, which is set to 10,000 microseconds.

4) Access the interface.

The interface can now be called with "SET" actions to alter the values of EPICS records, and the Sdp named "PARA" can be integrated to determine the value of EPICS records. Note that all calls to SET expect the argument value to be a string (?).

Example 1 set a value in an EPICS record

```
DitsArgType id;
ArgNew(&id,status);
ArgPutString(id,"Argument1","EPICS_record_name",&transid,status)
ArgPutString(id,"Argument2","10.4",&transid,status)
```

Example 2 get a value from an EPICS record

```
/* part_1() */
DitsGetParam(edifPath,"PARA",&transid, status);
/* - reschedule and await a return message - */
DitsPutObeyHandler(part_2,status);

/* part_2() */
DitsGetReason( &reason, &resonstat, status);
if ( reason == DITS_REA_COMPLETE ){
   double my_data;
   ArgID = DitsGetArgument();
   ArgGetd(ArgID, "EPICS_record_name", &my_data, status)
```

In version 1.0 of Edif, no record name aliasing is done, i.e. the EPICS record names are in the form "mch:cc:apply.DIR". Also, I think you HAVE to action "RELD" for "PARA" to be created.

5) Resetting the interface.
"RELD" (same semantics as "INIT") can be used at any time to the reinitialise which EPICS records to access. Since you can only add to the list of EPICS records looked at via "RELD", it is impossible to delete an EPICS record that is needed by another Drama task.

6) Exiting the interface.
When the interface is no longer needed, and action of "EXIT" can be sent to end the interface program. Just remember that Drama doesn't like tasks that disappear and reappear!

References

1. Interface Control Document: Mechanism Drama server task to EPICS OMC mechanism control module, **A. B. Gentles**, 29/11/96, Document number WHT-NAOMI-xxx.
2. Notes on NAOMI – ICS requirements, **Andy Longmore**, **Richard Myers**, 1/10/97, Document number AOW/SOF/AJL/2.0/10/97.
3. EDIF: A Prototype of EPICS/Drama interface, **Min Tan,** 11/98 (Discussion notes).
4. Design proposal for the command/status protocol and initial command set for the ELECTRA command sequencer**, R.M. Myers**, **D.F. Buscher**, **P.W. Morris**, 2/11/97. (see http://elsparc.dur.ac.uk)
5. Description of the CMD library, **P. W. Morris**, 20/5/97. (see http://elsparc.dur.ac.uk)
6. Description of the control of remote programs via BillDoors, **Patrick Morris,** 3/10/97. (see http://elsparc.dur.ac.uk)
7. Guide to Writing Drama Tasks, **Tony Farell**, 5/8/93, pp 31 – 37
8. ICD for NAOMI to TCS, **B D Kelly, 16/12/98**
9. ICD for NAOMI to Science Instrument (INGRID) , **B D Kelly, 16/12/98**