E U R O P E A N   S O U T H E R N   O B S E R V A T O R Y

Organisation Européenne pour des Recherches Astronomiques dans l'Hémisphère Austral

Europäische Organisation für astronomische Forschung in der südlichen Hemisphäre

VERY LARGE TELESCOPE

ESO New General Detector Controller

Infrared Detector Control Software

Design Description

Doc.No. VLT-SPE-ESO-13660-3837

Issue 2.5

Date 25/05/07

Prepared	J.Stegmeier	25/05/07	
	Name	Date	Signature
Approved	G.Finger		
	Name	Date	Signature
Released	A.Moorwood		
	Name	Date	Signature

Change Record

Issue/Rev.	Date	Section/Page affected	Reason/Initiation/Document/Remarks
0.1	30/11/05	All	First preparation.
1.0	27/03/06	2	Restructured. Added section for test software.
		3	Added sections for logging and error handling.
		4.6	The users access rights for the data-path are checked before the exposure is started.
		4.8	Added post-processing control loops.
		4.9	Added acquisition process continuous mode.
		3.1 to 3.4	Order has been changed. Voltage telemetry check added.
		3.4	Sytsem can be configured for pure hardware control as well as for pure data acquisition.
		3.7	Added directory tree.
		3.9	Configuration modules.
		6	Added traceability matrix.
		7	Updated.
2.0	10/08/06	All	Revised version.
2.1	30/08/06	3.1 7	New database architecture. Updated man-pages.
2.2	03/01/07	All 7	Aligned with ne naming conventins. Updated man-pages and CDT.
2.3	19/02/07	5 7	Updated panel screenshots. Updated database, man-pages and CDT.
2.4	19/02/07	5	Updated panel screenshots.
2.5	25/05/07	7	Updated man-pages.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Purpose

The document describes the design of the detector control software for infrared applications using the ESO New General detector Controller (NGC). It addresses to the hardware developer in the lab, to the detector specialist, to all instrument users including those of external consortia and to all software developers interfacing to the infrared detector control software.

1.2 Scope

The new general detector controller will be used for both optical and infrared applications. The NGC base software [AD9] covers all software functionality which does not yet impose any restriction in what will actually be done with the NGC. The software will become infrared and optical application specific when entering the data acquisition and when running exposure loops. Additionally some optical and infrared application specific devices need to be supported by software (shutter, chopper). So in the end there will be two specific control servers. This also implies the design of two different graphical user interfaces and two database branches. Nevertheless unnecessary differences should be avoided.

This document describes the server and GUI implementation for infrared applications. The possibility to scale down the control server for pure hardware control gives the oppotunity to use the server as a core process also in other applications, which then can apply different data taking and exposure mechanisms. In that sense also the associated engineering GUI may be used for common purposes.

The setup keywords will have a common root (the NGC dictionary in the software module “***dicNGC***”). Specific extensions will be described in additional dictionaries (TBD). The keyword names mentioned in this document are based on a first draft of the common dictionary, but they may still be subject to changes.

1.3 Applicable and Reference Documents

All applicable and reference documents are listed in the “NGC Project Documentation” document, VLT-LIS-ESO-13660-3906.

1.4 Glossary

See NGC Project Glossary [AD63].

1.5 Abbreviations and Acronyms

See NGC Project Acronyms [AD64].

2 OVERVIEW

2.1 System Architecture

The NGC infrared detector control software (***NGCIRSW***) is running partly on the instrument workstation (IWS) and on the NGC-LCU, where the physical interface(s) to the NGC detector front end reside. The NGC-LCU is a PC running a Linux operating system (kernel 2.4 or higher). All system communication is done via the control server. The control server is configured through configuration files or setup commands. The commands are received via the CCS message system. Image data is written to FITS files. Internal server data (status, current setup, ...) is mirrored in a database. Setup- and control-commands can be sent from a graphical user interface (GUI), which reads back the server settings from the database. Image data is visualized in one or more real time displays (RTD), which directly connect to the acquisition processes (see [AD9] for the data transfer mechanism) or may also read the data from the FITS file created by the control server. File creation is communicated through database events. The image data is received asynchronously from the acquisition processes.

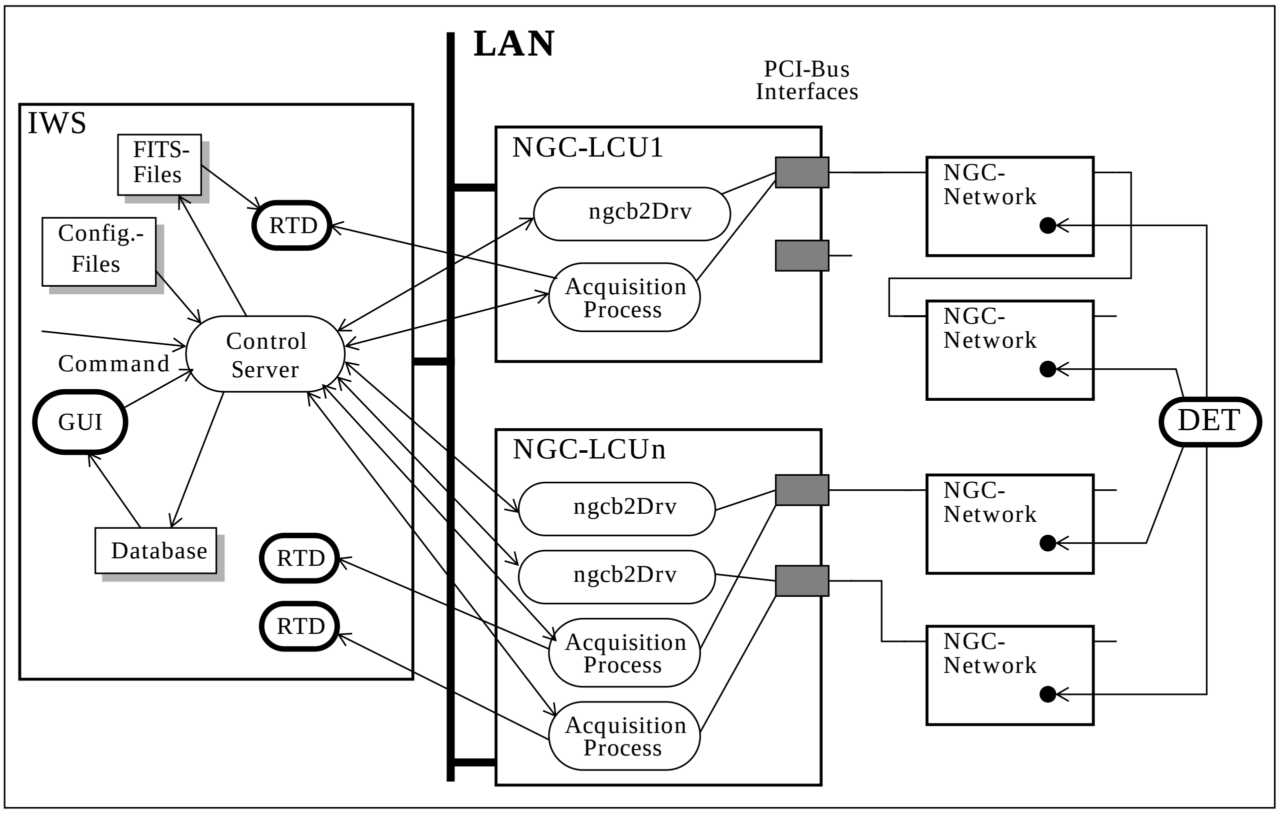


Figure 1 System Architecture

2.2 Processes

The control server communicates with the NGC hardware through driver interface processes (***ngcb2Drv***) running locally on the NGC-LCUs where the physical interfaces reside. The exact mechanism is described in [AD9]. One driver interface process is launched per physical interface device. The control server creates one ***ngcbIFC*** interface instance per physical device. In case the control server is not running locally on the NGC-LCU, the interface would include the driver interace process (using the derived ***ngcbIFC_MSG*** class instead of ***ngcbIFC***) as it is shown in Figure 2.

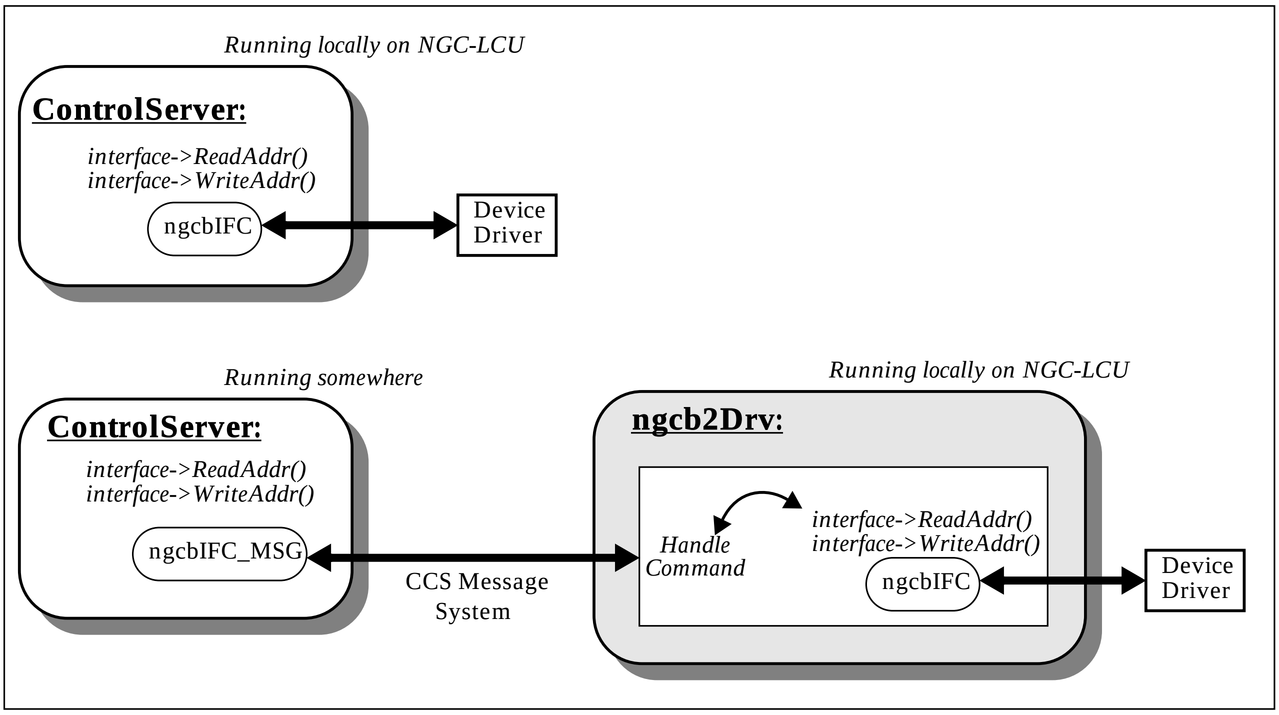


Figure 2 Control Server and Driver Interface Process

The acquisition processes are also launched and controlled by the control server. One ***ngcdcsACQ*** class instance (see [AD9]) per process is created in the control server to build the command interface and the asynchronous data interface. Nevertheless this does not imply that a process is actually running “on” this module in all operational modes. So it may happen that one or more of those ***ngcdcsACQ*** instances remain in an “idle” state during an exposure.

For maintenance and development operations all processes shown on the IWS side may run also locally on one of the NGC-LCUs. For software testing and software development all processes may run in simulation mode on the IWS.

In normal operation the driver interface process(es) on the NGC-LCU and the acquisition processes are allowed to receive commands only from the control server. There is no direct interaction foreseen between these processes and any higher level software except the data connection between the RTD and the acquisition process. For debugging purposes and software development it is possible to send commands directly to all processes (TBD).

The execution of the control processes normally does not require any special user privileges to be granted by the operating system. However the acquisition processes on the NGC-LCU benefit from the real-time capabilities granted by the Linux operating system to processes with super-user privileges.

2.3 Software Modules

All software modules are under CMM configuration control.

- ***ngcdrv*** - The device driver for the PCI-Bus back-end card.
- ***ngcb*** - The NGC basic software module containing the driver interface library (***ngcbDrv***) for communication and DMA, some basic i/o tools, a portable threads- and priority-control implementation and the C++ base classes for general system access. This module also provides a hardware simulation mechanism for the NGC controller.
- ***ngcpp*** - The DMA data-acquisition and pre-/processing module.
- ***ngcdcs*** - The NGC detector control software base module implementing the classes for the NGC hardware modules (sequencer, CLDC, ADC) and the interfaces to the data acquisition.
- ***ngcircon*** - The NGC system coordination module for infrared applications. This includes the infrared control server instance and all additional infrared specific device classes (chopper, special ADC or CLDC hardware releases, etc.). It also contains all required scripts for system startup and shutdown.
- ***ngc[ir?]gui*** - An engineering GUI used for direct system interaction and data acquisition. It could also be part of either the ***ngcircon*** or ***ngcdcs*** module (TBD).

A dictionary, which is common to both infrared and optical systems, is stored in the ***dicNGC*** software module.

The software module ***ngcarch*** provides automatic installation procedures for all the mentioned software modules.

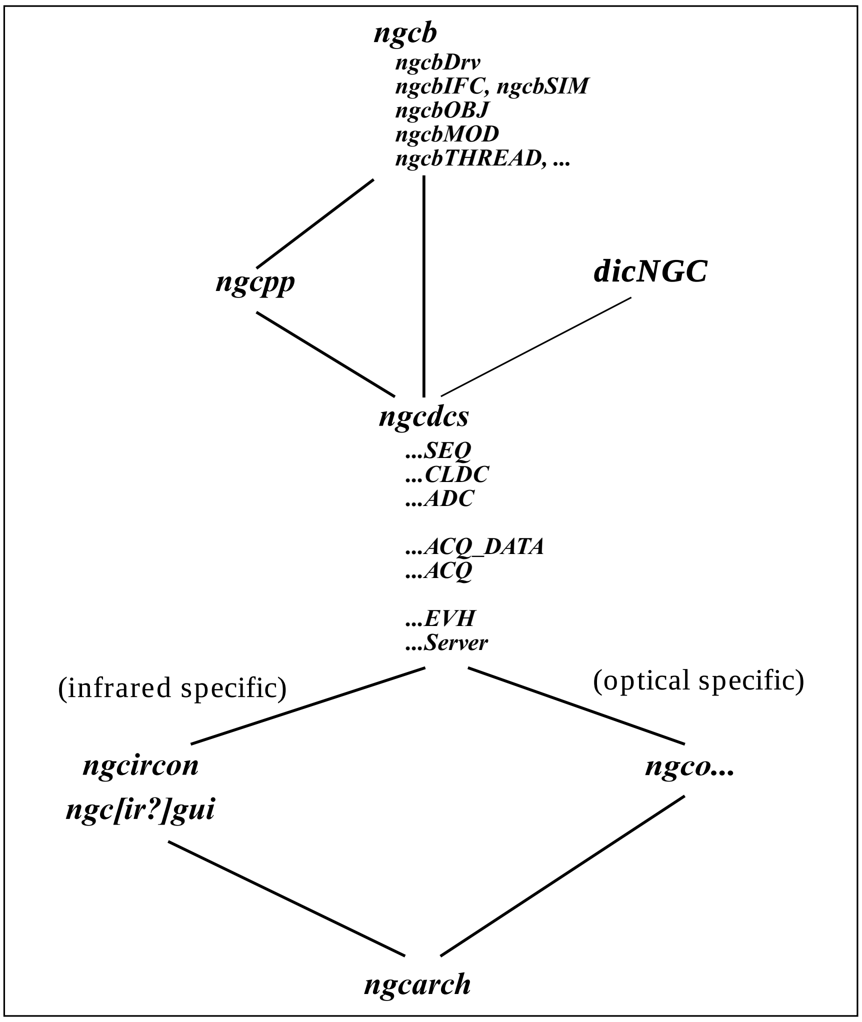


Figure 3 Module Dependencies

2.4 Test Software

Test scripts for the TAT (see [RD41]) are developed in parallel to the software module code generation. Test configuration files are created for various virtual system architectures and detector assemblies to cover all possible ranges of complexity. The DMA data-acquisition and pre-/processing module ***ngcpp*** does not contain TAT test scripts, but fully working test/template acquisition processes instead, which are then embedded in the test scripts and test configuration files of the higher level ***ngcdcs*** and ***ngcircon*** modules.

3 CONTROL SERVER

The control server is based on the CCS Event Tool Kit EVH (see [RD33], [RD35]). A base class (***ngcdcsEVH***) and a basic engineering server instance (***ngcdcsEvh*** - see section 7.2.1) are part of the ***ngcdcs*** module. The infrared control server instance (***ngcircon***) derives from the ***ngcdcsEVH*** class and introduces additional, specific behavior.

Several instances of the control server may run within the same environment. In this case an instance label (string) must be passed via the "*-inst*" command line option to distinguish between the systems. The instance label is used as appendix "***_<label>***" for both the database branch (see section 3.1) and for the server process name registered with the CCS environment.

All communication with the control server is done via the CCS message system and the online database. Any process in the VLT environment, which is able use to this communication structure, can directly interface to the system. In particular this involves the execution of test templates via BOB and the VLT control software HOS sequencer.

3.1 Database

Database classes for the controller modules and for the data acquisition are defined in the ***ngcdcs*** module (see section 7.1). This also contains base classes for exposure definition and for the overall system status (state, sub-state, system parameter-table, etc.).

The infrared detector control and data acquisition server (***ngcircon***) uses a database class (***ngcircon-SERVER.class***), which puts all these classes together. The file ***ngcircon.db*** finally contains the database branch definition. This file has to be included in the ***DATABASE.db*** file of the environment. The following macros can be defined before each inclusion:

```
#define ngcirconINSTANCE ngcircon_myInstance
#define ngcirconROOT :Appl_data:...:myPoint
#include "ngcircon.db"
```

ngcirconINSTANCE becomes the alias of the database point for this branch. The appendix <*myInstance*> should be the instance label as passed to the server with the "*-inst*" command line option. If not defined, ***ngcirconINSTANCE*** defaults to "***ngcircon***" (which is used by the server when setting no instance label). ***ngcirconROOT*** is the absolute path of the database root. If not defined it defaults to <***ngcirconINSTANCE***>.

The basic structure of the database is as follows:

```
--o <alias>ngcirconINSTANCE --|--o system     (NGC system parameters)
                              |--o exposure   (exposure parameters)
                              |--o mode       (read-out mode parameters)
                              |--o guiding    (guiding mode parameters)
                              |--o chopper    (chopper interface)
                              |--o seq_<i>    (sequencer parameters)
                              |--o cldc_<i>   (CLDC parameters)
                              |--o adc_<i>    (ADC module parameters)
                              |--o acq_<i>    (acquisition module parameters)
```

The branches for the sequencer-, CLDC-, ADC-, and acquisition classes are indexed. One branch will be created per module.

3.2 Server States

The server state is called "***OFF***" when no server process is running. Starting the server initializes the state to "***LOADED***". Then it is possible to send further commands to the control server. If a default detector configuration was specified in the system configuration, the detector configuration is loaded into the server. The command ***STANDBY*** brings the server to the "***STANDBY***"-state. If the server is not running locally on the NGC workstation, a driver interface process is started locally on the NGC workstation. The acquisition process is not yet running. The command ***ONLINE*** opens the connection to the physical device(s) and configures all modules according to the current system- and detector-configuration and also launches the acquisition process, if one was defined ("***ONLINE***"-state). The voltage telemetry of all CLDC instances is automatically checked before going to "***ONLINE***"-state. In "***ONLINE***" state the voltage telemetry can be checked at all times. Before going to "***ONLINE***"-state the SW reads all relevant product information (serial number, product code, revision number) from the hardware. This information is used to check the actual system configuration for consistency and to introduce revision specific behavior. It is possible to go directly from "***LOADED***"- to "***ONLINE***"-state and vice-versa (with the command ***OFF***). When going from "***ONLINE***" to "***STANDBY***" the acquisition process terminates, the sequencer is stopped, the CLDC voltage outputs are all disabled and the connection to the physical device is closed. The ***OFF*** command additionally terminates the driver interface process.

The detector front-end can be reset using the ***RESET*** command. This can be done in any state except "***OFF***". The acquisition process will always be stopped in this case.

The system is set to "***OFF***"-state when an ***EXIT*** command is received.

The server sub-state is "***busy***" during command execution. In case the server was not able to recover by itself from an error, the sub-state is "***error***" (i.e. the server is in error state). While an exposure is running, the sub-state is "***active***". In all other cases the sub-state is "***idle***".

The server state is stored in the database attributes '***<alias>ngcircon:system.state/subState***'. A translation of the state numbers into a string value is stored in '***<alias>ngcircon:system.stateName/subStateName***'.

3.3 Verbose Mode and Logging

Verbose messages can be printed on the standard output stream of each process. The detail is given by a verbose level, which is also passed as parameter to the control server ("*-v <level>*" command line option). To make the messages of the sub-processes visible, it is required to start those processes in a separate terminal (this is controlled with the "*-xterm*" command line option).

Error - logging will be done with the standard CCS error logging facility, which includes the automatic logs like tracing of any received/sent command (see [AD27], [RD32]). Additionally the verbose output can be logged in a detail depending on a given log-level for maintenance/debugging purposes. Operational logs are TBD.

3.4 Error Handling

The CCS error mechanism [RD32] provides a classification scheme for application specific errors. The NGC base software uses this mechanism. The introduction of new error codes is limited to cases, where specific actions ("*reset*", "*restart server*", "*restart CCS environment*", "*reboot*" etc.) are required. Other errors, which leave the system still in a valid state without further interaction ("*parameter out of range*", "*invalid file name*",...) are trapped by an overall system error (***ngcbERR_SYSTEM***, ***ngcdcsERR_SYSTEM***) plus an appropriate message string. The meaning of the error class and the possibly needed interactions are described in a help file (.hlp), which can be displayed with the standard CCS-tools (also with the *logMonitor*). The actual error reason (*"timeout"*, *"link channel error"*,...) is given in an associated error message string. A draft list of possible error classes is given in [AD9].

3.5 Configuration

The overall configuration is divided into controller electronics system configuration and detector configuration. The controller electronics system configuration describes the hardware system used. The detector configuration describes the usage of the system with respect to the connected detector(s). There are cases, where more than one detector is driven by the same hardware and the switch between the detectors has to be done by applying a different detector configuration (i.e. enable / disable a different set of CLDC- and / or ADC-modules). To reflect such cases, where different detector configurations are used on the same system configuration (or vice-versa the same detector configuration is used on different system configurations), the two files have to be kept separated in order to avoid to unnecessarily duplicate the information.

3.5.1 Controller Electronics System Configuration

The system configuration is given in a configuration file (short FITS format). The file can be specified via the "***-cfg***" command line option of the control server. A new system configuration file can be loaded at server run-time via a setup command (***DET.SYSCFG*** *<filename>*). The file content is translated by an upload-method into a structure of type *ngcdcs_syscfg_t*. A *CheckSysCfg(ngcdcs_syscfg_t cfg)* member function will check ranges and consistency of the members of such a structure. The structure is applied via the *Configure(ngcdcs_syscfg_t cfg)* method, which also foresees two call-backs to configure additional stuff before and after internal configuration is done. So it is not mandatory to fill the structure by loading a file as it is given in the below example. The configuration may still be loaded from an on-line database or whatever file format by adding a special uploading function or changing the default one. However the loading of such a keyword based file has been proven to be robust and fail-safe in the past and it is also easy to be inspected by view in its raw loadable format. An editing tool (***ngc[ir]guiCfg***) for the system configuration is provided by the ***ngc[ir]gui*** module (see section 5.3).

The system configuration includes all information to identify the hardware configuration including the interface device names and the computing architecture (host names, environments,...). Here the controller interfaces are defined and associated to the linear list of hardware modules. Each device is declared via a block of keywords giving the device name (***DET.DEVi.NAME***), the host name (***DET.DEVi.HOST***), where the physical interface resides, and the name of the CCS environment (***DET.DEVi.ENV***) running on this host. If no host name is specified (empty string), the interface is assumed to be on the same computer where also the control server is running. In this case no additional driver interface process will be launched and the environment name is ignored (like the optional "driver interface process" name in ***DET.DEVi.SRV***). If the host name is set to the local host name ($HOST), the driver interface process would be started even if it was not really needed. This is used for testing the software when running in simulation mode on a single workstation. Finally an optional device type (***DET.DEVi.TYPE***) can be given in order to use other interface devices derived from the ***ngcbIFC*** class.

The NGC hardware modules (sequencer, CLDC, ADC) are realized in SW also in a modular way (C++ classes ***ngcdcsCLDC***, ***ngcdcsSEQ***, ***ngcdcsADC***, - see [AD9]). One class instances of the respective type is created for each hardware module in the NGC detector front end. The interface object (as specified via the ***DET.DEVi*** keywords) is given as parameter to the module class instances together with the linking route *through* the interface *to* the target module. So finally the server will just see linear lists of sequencer-, CLDC- and ADC-modules independent from the nesting structure of the NGC hardware module network(s).

Each of the hardware modules (SEQ, CLDC, ADC) gets one interface device assigned, through which it is accessed. The assignment is done via a reference index in the ***DET.SEQi.DEVIDX***, ***DET.CLDCi.DEVIDX***, ***DET.ADCi.DEVIDX*** keywords. For each of the modules an optional name can be defined.

The acquisition modules also have to be declared here. For each DMA device one module needs to be defined. This does not yet define the actual process, which will be launched by the module. The modules are attached to individual sequencer instances with the ***DET.ACQi.SEQIDX*** keyword. Each module refers to exactly one sequencer, which "produces" the detector data received by it. Several acquisition modules may be associated to the same sequencer instance. This association is needed to have the information, when the processes need to be (re-)started or stopped. If no index is defined a default value of 1 (first sequencer) will be used. The negative index (-1) tells the system not to associate the acquisition module to any sequencer.

There is no restriction in the number of hardware modules and the number of acquisition modules. So the system may be configured for pure hardware control (no acquisition modules) or even for pure data acquisition (no controller interface device, no hardware module).

In case several sequencer modules are in the system, it will be possible to start/stop them (plus the process on the associated acquisition modules) individually or all at once (synchronously). If several CLDC modules are in the system, it would likewise be possible to enable/disable them individually or all at once. But here there is the restriction, that a (checked) voltage configuration file must have been loaded into such a CLDC module, before it can be enabled automatically with an "enable all" command.

The controller electronics system configuration parameters are described in a dictionary [AD37]. Before going to "***ONLINE***"-state the actual hardware is checked against this configuration and an error is reported in case something does not match.

Example of System Configuration File:

```
# Server configuration
DET.DETCFG         “test.dcf”;      # default detector configuration

# Exposure frame configuration
DET.FRAM.FORMAT    “extension”;     # default FITS-file format
DET.FRAM.MULTFILE  F;               # generate multiple files
DET.FRAM.NAMING    “request”;       # default FITS-file naming scheme

# Device description
DET.DEV1.NAME      “/dev/ngc0_com”; # associated device name
DET.DEV1.HOST      ““;              # host where interface resides
DET.DEV1.ENV       “$RTAPENV”;      # server environment name
DET.DEV1.SRV       ““;              # optional server name
DET.DEV1.TYPE      ““;              # optional type

# CLDC modules
DET.CLDC1.DEVIDX   1;               # associated device index
DET.CLDC1.ROUTE    “2”;             # route to module
DET.CLDC1.AUTOENA  T;               # auto-enable at online
DET.CLDC1.MARGIN   0.2;             # margin for voltage check (in volts)
DET.CLDC1.NAME     “CLDC 1”;        # optional name
DET.CLDC2.DEVIDX   1;               # associated device index
DET.CLDC2.ROUTE    “5,2”;           # route to module
DET.CLDC2.AUTOENA  T;               # auto-enable at online
DET.CLDC2.MARGIN   0.2;             # margin for voltage check (in volts)
DET.CLDC2.NAME     “CLDC 2”;        # optional name

# Sequencer modules
DET.SEQ1.DEVIDX    1;               # associated device index
DET.SEQ1.ROUTE     “2”;             # route to module
DET.SEQ1.NAME      “Sequencer 1”;   # optional name
DET.SEQ2.DEVIDX    1;               # associated device index
DET.SEQ2.ROUTE     “5,2”;           # route to module
DET.SEQ2.NAME      “Sequencer 2”;   # optional name

# ADC modules
DET.ADC1.DEVIDX    1;               # associated device index
DET.ADC1.ROUTE     “2”;             # route to module
DET.ADC1.NUM       4;               # number of enabled ADC units on board
DET.ADC1.BITPIX   18;               # number of bits per pixel
DET.ADC1.FIRST     T;               # first in chain
DET.ADC1.PKTCNT    1;               # packet routing length
DET.ADC1.NAME     “ADC-Module 1”;   # optional name
DET.ADC2.DEVIDX    1;               # associated device index
DET.ADC2.ROUTE     “5,2”;           # route to module
DET.ADC2.NUM       4;               # number of enabled ADC units on board
DET.ADC2.FIRST     F;               # first in chain
DET.ADC2.PKTCNT    0;               # packet routing length
DET.ADC2.NAME     “ADC-Module 2”;   # optional name

# Acquisition modules
DET.ACQ1.DEV       “/dev/ngc0_dma”; # DMA device name
DET.ACQ1.HOST      “$HOST”;         # host name for acq.-process
DET.ACQ1.CMDPORT   0;               # acq.-process command port (optional)
DET.ACQ1.DATAPORT  0;               # acq.-process data port (optional)
DET.ACQ1.NCLIENT   2;               # max. number of data server clients
DET.ACQ1.SEQIDX    1;               # associated sequencer instance
DET.ACQ2.DEV       “/dev/ngc1_dma”; # DMA device name
DET.ACQ2.HOST      “$HOST”;         # host name for acq.-process
DET.ACQ2.CMDPORT   0;               # acq.-process command port (optional)
DET.ACQ2.DATAPORT  0;               # acq.-process data port (optional)
DET.ACQ2.NCLIENT   2;               # max. number of data server clients
DET.ACQ2.SEQIDX    1;               # associated sequencer instance
```

3.5.2 Detector Configuration

Once the system is basically configured, a certain configuration for a detector (or a detector mosaic) needs to be applied. Similar to the controller electronics system configuration the detector configuration is also stored in a configuration file (short FITS format), which can be loaded at server run-time via a setup command (***DET.DETCFG*** *<filename>*). The file content is translated by an upload method into a structure of type *ngcdcs_detcfg_t*. If the system is in online state, the configuration is directly applied to the hardware. Otherwise a preset is done, which is applied when going online at a later time.

In the detector configuration file the chips used in this setup are defined. The chips get a name, an id, a type and some more information (like position/gaps in a mosaic) assigned. Most of these (***DET.CHIPi.XXXX***) keywords are just forwarded to the FITS-file header. The ***DET.CHIPi.NX/NY*** keywords define the detector dimension used as a basis for all images and windows. The ***DET.CHIPS*** keyword defines the number of chips in a mosaic (default value is "1"). There is also a keyword to assign this chip to a certain acquisition module (***DET.CHIPi.ACQIDX***). This is required to pass the right frame dimensions and window parameters to the associated acquisition process. A zero index means that the chip definition applies to all acquisition modules.

The detector configuration file defines the clock pattern configuration files, which have to be loaded into the sequencer module(s) for this detector/mosaic (***DET.SEQi.CLKFILE***), and also the default values for the global state dwelltime (***DET.SEQi.TIMEFAC/DET.SEQi.TIMEADD***) and the "*continuous mode*" flag (***DET.SEQi.CONT***). The ***DET.SEQi.RUNCTRL*** keywords give the information which sequencer instances will be started synchronously (i.e. will react on the external run-signal). The detector voltage configuration files (***DET.CLDCi.FILE***) have to be specified for all CLDC modules which are used in this detector configuration. Not used CLDC modules can simply be skipped and their outputs will not be automatically enabled together with the other ones.

For each ADC module declared in the system configuration some keywords can be entered to tune the A/D conversion (delays, offsets), to set the modules to various simulation modes or to enable/disable groups of ADCs on this module.

The detector configuration also declares the read-out modes to be used with the given chip(s). Each read-out mode is defined by a block of keywords assigning the name of the mode (***DET.READi.NAME***), specifying the sequencer program(s) to be loaded (***DET.READi.SEQi***), the acquisition processe(s) to be launched on the defined acquisition modules (***DET.READi.ACQi***), a default parameter setup file (short FITS format) to be loaded, whenever the mode (***DET.READi.DSUP***) is selected, and a short description string (***DET.READi.DESC***). A default read-out mode can be specified by its index (as given in ***DET.READi***). The default read-out mode is applied, whenever the file is loaded respectively when the server switches to online state after the file has been loaded.

The detector configuration parameters are described in a dictionary [8].

Example of Detector Configuration File:

```
# Detector system definition
DET.NAME    “myName”;    # detector system name
DET.ID      “myId”;      # detector system id
DET.CHIPS     1;         # number of chips in mosaic

# Chip definition
DET.CHIP1.NAME     “myChipName”;  # chip name
DET.CHIP1.ID       “myChipId”;    # chip id
DET.CHIP1.TYPE     “myChipType”;  # chip type
DET.CHIP1.DATE     “2005-08-03”;  # chip installation date
DET.CHIP1.LIVE     T;             # chip live or broken
DET.CHIP1.PXSPACE  1.0E-06;       # space between pixels (meters)
DET.CHIP1.PSZX       1.0;         # size of pixel in x (mu)
DET.CHIP1.PSZY       1.0;         # size of pixel in y (mu)
DET.CHIP1.INDEX       1;          # unique number in mosaic
DET.CHIP1.X           1;          # x location in mosaic
DET.CHIP1.Y           1;          # y location in mosaic
DET.CHIP1.XGAP        0;          # gap between chips along x
DET.CHIP1.YGAP        0;          # gap between chips along y
DET.CHIP1.RGAP        0.0;        # angle of gap between chips
DET.CHIP1.OUTPUTS    32;          # number of outputs
DET.CHIP1.NX       1024;          # number of pixels along x
DET.CHIP1.NY       1024;          # number of pixels along y
DET.CHIP1.ADJUST  "FREE";         # window adjustment (CENTER|FREE)
DET.CHIP1.ADJUSTX     1;          # adjustment step in x
DET.CHIP1.ADJUSTY     1;          # adjustment step in y
DET.CHIP1.ACQIDX      0;          # map to acquisition module

# CLDC module setup
DET.CLDC1.FILE     “test.v”;      # voltage definition file
DET.CLDC2.FILE     “test.v”;      # voltage definition file

# Sequencer module setup
DET.SEQ1.CLKFILE   “test.clk”;    # clock pattern file
DET.SEQ1.TIMEFAC   2;             # dwell time factor
DET.SEQ1.CONT      F;             # continuous mode
DET.SEQ1.RUNCTRL   T;             # external run-control

# ADC module setup
DET.ADC1.DELAY      0;            # conversion strobe delay (ticks)
DET.ADC1.ENABLE     4;            # number of enabled ADC units
DET.ADC1.OPMODE     0;            # ADC operation mode
DET.ADC1.SIMMODE    0;            # ADC simulaton level
DET.ADC1.PKTSIZE    2;            # packet size
DET.ADC1.CONVERT1   T;            # convert on strobe 1
DET.ADC1.CONVERT2   F;            # convert on strobe 2

DET.READ.DEFAULT    2;           # id of default readout mode

# Read-out mode definitions
DET.READ1.NAME   “Uncorr”;          # readout mode name
DET.READ1.ACQ1   “ngcppSimple16”;   # acquisition process on module 1
DET.READ1.ACQ2   “ngcppSimple16”;   # acquisition process on module 2
DET.READ1.SEQ1   “test.seq”;        # program for sequencer 1
DET.READ1.DSUP   “”;                # default parameter setup
DET.READ1.DESC   “uncorrelated readout”;

DET.READ2.NAME  “Double”;          # readout mode name
DET.READ2.ACQ2  “ngcppTemplate”;   # acquisition process 2
DET.READ2.SEQ1  “test.seq”;        # program for sequencer 1
DET.READ2.DSUP  “”;                # default parameter setup
DET.READ2.DESC  “double correlated readout”;
```

3.5.3 Directory Tree

Unless an absolute path is defined, all file locations are relative to the location of the file they are referred from. So the parent directory of the actually loaded system configuration file would become the reference path for the referred detector configuration file and for the parameter default setup file in the system configuration scope. The parent directory of the actually loaded detector configuration file will become the reference path for the referred voltage setup files, for the clock pattern configuration files, the sequencer programs and also for the parameter default setup files in the detector configuration scope.

This scheme basically introduces one main entry point (the directory of the system configuration file). All other paths are derived from there and so the directory tree is freely adaptable through the filename entries themselves (i.e. "*../SEQ/myProgram.seq*"). The default search path for the system configuration file is "*$INS_ROOT/SYSTEM/COMMON/CONFIGFILES/*". The ***ngcircon*** server introduces a default system configuration file ("*NGCIRSW/ngc.cfg*"), which is used in case nothing had been specified via the "***-cfg***" server command line option, whereas the ***ngcdcsEvh*** engineering server uses a built-in default configuration in this case (i.e. one main board + one acquisition module).

3.5.4 Configuration Modules

The configuration files and sequencer programs are stored in instrument- or detector-specific configuration modules, which are under CMM configuration control. This also applies to maintenance and test configurations. Usually the configuration files are part of the NGC-system delivery. In cases where detector development is done outside, the respective instrumentation team is responsible for the development of the configuration files. Templates and a graphical editing tool for the clock patterns will be provided.

3.6 Simulation Mode

The default operation mode is the so-called "***NORMAL***" mode, where all physical devices are accessed. The default mode can be changed with the ***DET.CON.DFEMODE*** keyword in the system configuration file. The value can be either "***NORMAL***" or "***HW-SIM***" or "***LCU-SIM***". The keyword is overwritten by the "***-sim***" server command line option, which would force the server to start in the given simulation mode (HW or LCU) in any case. At server run-time the operation mode can be changed with the commands "***SIMULAT*** -function HW|LCU" and ***STOPSIM***.

In HW-simulation mode the NGC hardware is simulated using the ***ngcbSIM*** simulator instances in the interface devices. The acquisition process is then also started in data simulation mode. The data timing (simulation interval) is set according to the current detector integration time (DIT). More realistic timing simulation may be ***TBD***. The simulation can be started on a single workstation by setting the host names for the acquisition process(es) and for the interface devic(es) to the local host. In this case the launch of the driver interface process can also be skipped by giving an empty string as interface device host name. The interface would then be created as "local" interface.

The LCU-simulation mode additionally forces all processes to be started on the hosting workstation (IWS).

The current operation mode is stored in the database attribute '***<alias>ngcircon:system.opMode***'.

For software testing it is possible to emulate various errors:

- No acknowledge from hardware module (“*no_ack*”)
- Invalid address on a NGC hardware module (“*ivld*”)
- Sequencer FIFO empty (“*seq_empty*”)
- Sequencer goes to IDLE state (“*seq_idle*”)
- I/O error on all communication links (“*io_err1,2,3*”)
- I/O error on data link (“*data_io*”)
- Error when writing data to a file (“*data_file*”)
- Server blocks forever (“*block*”)

Others may be added. The emulation can be done with the low-level maintenance command “*NGC simerr <identifier>*” (see 7.2.2). A specific command for this purpose and a more convenient identifier assignment is TBD.

3.7 Parameter Setup

Parameters are sent via the command

“***SETUP -function <name1> <value1> [<name2> <value2>...]***”

The order of the parameters within such a setup command is insignificant. The parameter names typically start with “***DET.***”. The ***DET*** category may be indexed. The value of a parameter can be retrieved with the command

“***STATUS -function <name1> <value1> [<name2> <value2>...]***”

The parameter values are returned as a list in the same order as given in the command:

<name1> <value1>[, <name2> <value2>,...]

String values are enclosed in double quotes. The command may return intermediate replies when the list becomes too long.

Additionally to the common system parameters, the data pre-processing and the sequencer program may introduce an arbitrary number of application specific parameters. These parameters have a meaning only in their limited context, but nevertheless they may have an impact on the overall system, as they may require sequencer reloading or an acquisition process restart. In order not to do those time consuming functions too often, their system influence needs to be described somehow. To achieve this, the application specific parameters are stored in a parameter list, which for each element carries some flags for specific actions to be performed, when the parameter value changes (see [AD9]). The list also eases to keep an overview of all parameters in use (i.e. to visualize and to set them in a graphical user interface). Range and validity checks are done by the control server. Only after successful application of the parameter the value in the list will be updated accordingly. Now the problem arises how a certain parameter may enter this list:

Generally all parameters are described in one or more dictionaries (see [AD37]). Those dictionaries will contain both the common system parameters and the application specific ones but define rather all possible parameters, which *may* be set at some time, than those, which currently are really in use. Nevertheless any parameter, which may enter our list, must be defined in a dictionary in order to be visible to the outside world. The dictionary also defines, which of the parameters will be added to the FITS-headers of the produced data files.

The parameters used by an acquisition process can be directly imported from the process itself (see [AD9]). So the control server knows which process uses which parameters. The only thing to be done here is to ensure, that the parameters of all acquisition processes used by a certain detector configuration are declared in a dictionary. Nevertheless there may also be parameters, which are only used within a sequencer program, but not in any acquisition process. In order to add them to our list, they need to be declared with a default value in a parameter default setup file (.dsup). These setup files may be defined within the scope of the system configuration, of the detector configuration or of the read-out mode definition. This scope also defines a parameters "*life time*" in the list. A parameter declared at detector configuration level will be removed or reset to (a possibly new) default, when a new detector configuration is loaded. A parameter declared at system configuration level will be removed or reset to (a possibly new) default when a new system configuration is loaded. The command

"***SETUP -default***"

will apply all default values described in the parameter default setup files.

The dynamic parameter list is stored as a table (fields: *name*, *value*) in the database attribute '***<alias>ngcircon:system.param***'. It may also be retrieved by sending the ***STATUS*** command with no parameter.

3.8 Server Extensions

Server extensions can be built by deriving from the ***ngcdcsEVH*** class (see appendix section 7.2.3). Post-processing callbacks may be added as described in section 4.8. Hooks for special actions at server state switching (before and after "*online*", before and after "*standby*", before and after "*loaded*"), when setting parameters and upon exposure events may be installed. The system- and detector-configuration file loading functions *LoadSysCfg()* and *LoadDetCfg()* can be overloaded to support different file formats. The *ConfigureCB()* call-back function can be overloaded to install interfaces, which are derived from the ***ngcbIFC*** class and to configure parts of the system, which are not covered by the basic control server. Module creation hooks can be installed to create derived versions of the sequencer-, CLDC-, and ADC-modules and also of the acquisition module (see [AD9]). The modules are created when applying the system configuration with the *Configure()* method as described in section 3.2. The modules are deleted with the *Shutdown()* method, which is always called as first step in the *Configure()* method. The *Shutdown()* method provides two call-backs to shutdown application specific stuff before and after internal shutdown is done. Calling *Shutdown()* a second time in direct succession will always have no further effect.

3.9 Maintenance Mode

The maintenance service mode as described in [AD9] is also implemented in the ***ngcdcsEVH*** class, as this derives from the ***ngcdcsSRV*** class. The inheritance keeps a backwards compatibility for the test macros developed for testing the prototype hardware. The low-level service commands can be passed through via the ***NGC "cmd-string"*** command as defined in the command definition table (see section 7.2.2). The internal command shell for these functions can also still be launched with the *-shell* command line option. Once the control servers for infrared and optical applications are fully operational, this maintenance service mode may be removed (TBD).

4 EXPOSURES

4.1 Read-Out Modes

The read-out modes, which are defined in the detector configuration file (see section 3.5.2), can be selected at run-time with a setup command either by their name (***DET.READ.CURNAME***) or by their "*id*" (***DET.READ.CURID***). The setup usually requires the sequencer(s) to be stopped, reloaded and restarted, when the mode changes. The "*stop if running*" and "*restart if it has been running before*" mechanism will be executed automatically where required and is transparent to the caller of the set-up command. The mechanism may also be applied on other setup parameter changes, in case those parameters are used in the sequencer program.

The available read-out modes are stored as a table (fields: *name, id, desc*) in the database attribute '***<alias>ngcircon:mode.readModeList***'. The current read-out mode is stored in the attribute '***<alias>ngcircon:mode.readModeName/.readModeId***' (as integer reference id) and additionally in the attribute '***<alias>ngcircon:mode.readModeName/.readModeName***' (as string). A list of all available read-out modes can be retrieved with the command "***STATUS -function DET.READ.AVAIL***".

4.2 Frame Types

The application specific acquisition processes may produce an arbitrary number of frame types. Each frame type has two flags associated to define whether frames of that type will actually be produced by the pre-processor and whether frames of that type should be stored to disk during an exposure. A software window can be defined individually for each type and for each acquisition module using the ***FRAME*** command as described below.

Usually an exposure is finished, when the INT-frame has been received on the instrument workstation. As it is required by some read-out modes to store also other frames during one exposure, a more general exposure break condition has to be applied: each frame generated by the acquisition process and selected to be stored can have a counter, that indicates the number of frames of that type, that must be stored during the exposure. The exposure is finished, when all of these frames have reached their break condition. A break condition of zero means that frames of this type should be stored on a "best effort" basis (i.e. "*store as much as possible until the exposure is finished*"). If all break conditions are set to zero, the exposure will run and store frames until it is aborted. All that can be controlled via the command:

> "***FRAME [-module <acq.-module id>] -name <frame name> [-gen T|F] [-store T|F]***
> ***[-break <counter>] [-win sx sy nx ny]"***

The frame setup can be done "per process" by specifying an acquisition module id. If the module id is zero or if no module id is passed, the command will refer to "*all*" modules.

The available frames are stored as a table (fields: *name, generate, store, breakCond, sx, sy, nx, ny*) for each acquisition module in the data base attribute '***<alias>ngcircon:acq_<i>.frame***'. The frame list can also be received with the command "***STATUS -function DET.READ.FRAMES***".

4.3 Windows

The setup parameters ***DET.WIN.STRX***, ***DET.WIN.STRY***, ***DET.WIN.NX***, ***DET.WIN.NY*** define the format and position of the data-frame within the chip. ***DET.WIN.STRX/Y*** always refers to the lower left corner. The sequencer module will derive its default read-out window (***DET.SEQi.WIN.STRX/Y***, ***DET.SEQi.WIN.NX/Y***) from these parameters. For detector mosaics (***DET.CHIPS*** > 1, see section

4.7) the window is by default intended to be applied “per chip”. Nevertheless both sequencer program and the associated acquisition module(s) may use these parameters in an application specific way to read-out detector overlapping windows. Whether and how a read-out window is applied depends on the detector architecture. The acquisition process will always know, what it will get, and will then setup the right DMA and sort the data properly.

The window used by the acquisition process is usually overtaken from the ***DET.SEQi.WIN*** parameters of the sequencer, to which it is associated. When the read-out window changes, the sequencer program is reloaded with the new window parameters and the acquisition process is restarted with the ‘-nx, -ny’ command line options set accordingly. In some cases this rule might be impractical and the window for the acquisition process needs to be set independently. The ***DET.ACQi.WIN.STRX/Y***, ***DET.ACQi.WIN.NX/Y*** parameters are introduced for this purpose. Changes in the ***DET.SEQi.WIN*** parameters will always automatically change the corresponding ***DET.ACQi.WIN*** parameters of all associated acquisition processes, but not vice-versa.

Window read-outs can be disabled via a parameter, which should be flagged in the default parameter setup file for read-out modes which do not support a window read-out of the detector. The window setup parameters may automatically be adjusted following several detector specific rules which are defined in the detector configuration:

The ***DET.CHIP.ADJUST*** parameter specifies the read-out window adjustment mode for the detector. Valid values are:

```
“CENTER”   - window is automatically centered
“FREE”     - window is only adjusted to multiples of
             the DET.CHIP.STEPX,STEPY parameters
```

The ***DET.CHIP.STEPX/Y*** parameters specify the adjustment step in x/y-direction for window read-out. Adjustments are done in multiples of this value.

Multiple windows (i.e. regions of interest) can be read-out by applying a proper set of additional window parameters within the scope of the sequencer program (see [AD9] for details). This mechanism is fully application specific.

Software windows are applied on a per-frame basis (see section 4.2). In this case the data transfer task just requests a window from the acquisition process. This is mainly used to save transfer and storage overheads.

4.4 Burst Mode

In some cases it might be necessary to store larger amounts of raw data or to sample at a very high frame-rate. If the frame rate is too high (> 200 Hz on most non-real-time UNIX platforms) the DMA-interrupt latency becomes dominating and no more CPU-power is left for pre-processing. Two kinds of “burst modes” are used to cover these two cases.

4.4.1 Raw Data Mode

The raw data mode is activated by sending the setup command

“***SETUP -function DET.ACQi.BURST.NUM <num>***”

If ***num*** is greater than zero, it indicates the number of [***DET.ACQi.WIN.NX, DET.ACQi.WIN.NY***]-dimension sample-frames to be stored in the burst buffer. If an exposure is started both sequencer and acquisition are restarted (the ***DET.SEQi.CONT*** flag is ignored in that case) and the buffer is filled until ***num*** sample-frames (16 bit short integer) are stored. At the end of the exposure an INT-frame (containing only dummy data) is transferred. The transfer of the sample-frames starts immediately and runs in parallel to the data recording. A 3 X 2 MBytes input ring-buffer is used to ensure interrupt stability. This means that at least 2 MBytes of data have to be created to fill the DMA. The setup parameter ***DET.ACQi.BURST.SKIP*** indicates the number of frames to be skipped before starting to transfer.

This mode can be activated regardless of the currently selected read-out mode. Setting ***DET.ACQi.BURST.NUM*** to zero deactivates the burst-mode and restores the defined acquisition process for the current read-out mode. The default burst-process (*ngcppBurst*) applying the described mechanism may be changed by specifying a user-defined burst-process (***DET.ACQi.BURST.PROC***).

4.4.2 Internal Burst Mode

The internal burst is activated by sending the setup command

“***SETUP -function DET.ACQi.BURST.NUM <-num>***”

The negative value indicates that an internal burst-buffer should be applied. In this case the DMA is enlarged by a factor of ***num***. The data processing is not affected in this case as soft-interrupts are created for each sub-division. The calculation thread will wait for ***num*** buffers and will then process them within one step. This helps to workaround the 200 Hz interrupt limitation, but depending on the actual processing it may slow down the acquisition performance as a whole (if for example least-square fit or standard deviation have to be solved within the ***num*** buffers).

Also in this case a value of zero for ***DET.ACQi.BURST.NUM*** deactivates the burst-mode.

4.5 File Formats

There are several FITS file formats supported to cover various situations. The simplest case is, that all frames produced during one exposure are stored into individual files (***DET.FRAM.FORMAT = “single”***). This is mainly used for detector tests in the laboratory to have a fast and simple quick-look to the generated data files. In case many intermediate results are produced, the FITS-header creation for each individual frame may introduce large overheads in both transfer time and needed disk space. To overcome this, the frames may be stored into data-cubes (***DET.FRAM.FORMAT = “cube”***). One cube would be created per frame type. This is especially needed for storing data in burst mode, where usually only very small windows are read out. To store several thousands of those small windows in binary image extensions or even single files would imply an enormous overhead.

The standard file format would be to store the frames produced during one exposure into binary image extensions (***DET.FRAM.FORMAT = “extension”***). If the data are coming from different processes, they will normally be available all at the same time. When storing to different files (i.e. one FITS-file per acquisition process containing all frames delivered by this process), all transfer can be done in parallel and the transfer processes need not to wait for each other before saving data to disk (***DET.FRAM.MULTFILE = “T”***). In the right configuration this would improve the transfer performance considerably. Nevertheless, when transfer performance is not the limiting factor, the storage mechanism is configurable to have only one binary image extension file per exposure (***DET.FRAM.MULTFILE = “F”***). A default value for this can be defined in the controller electronics system configuration file. It must also be considered, that when storing data from different acquisition processes into the same file, then the order of the individual image extensions is **undefined**! In any case each image extension contains a unique identifier in the extension header.

In case of very long integrations it might be required to inspect some intermediate data and to check them for consistency while the exposure is still running in order to avoid loosing telescope time, when something went wrong. In such cases the "***extension***" format is not practical as the data on disk can first be accessed when the exposure has finished. The "***single***" file format may be chosen to allow such intermediate quick-looks. This may require some FILE-merging to be done by higher level SW (OS) before passing the data to the archive.

The information which system parameters (if used in the actual context - see section 3.7) will appear in the FITS header is defined in the dictionaries [AD37].

4.6 Data File Naming

Three different naming schemes are available for the files being produced during an exposure. Unless an absolute path name is specified in the (base-)name all files will be stored by default in the data-path ***$INS_ROOT/$INS_USER/DETDATA***. The user's access rights for the data-path are checked before the exposure is started.

The naming scheme is set by the ***DET.FRAM.NAMING*** keyword. The keyword can be set either via ***SETUP*** command or in the system configuration file. The value is one of "***request***" or "***sequence***" or "***auto***":

1. ***Request Naming***: The name must be specified before each exposure is started. The name is given in with the SETUP command (parameter "***DET.FRAM.FILENAME <name>***"). The file will be named in the following way:

   ```
   <name>[_<frame-name>][_<frame-number>].fits
   ```

2. ***Sequence Naming***: An index is added to a base name. The index is incremented after each exposure. Setting the base name is done with the SETUP command (parameter "***DET.FRAM.FILENAME <name>***"). The index can be set with the "***DET.FRAM.SEQIDX <no>***" parameter. The FITS-file will be named in the following way:

   ```
   <name><seq-index>[_<frame-name>][_<frame-number>].fits
   ```

3. ***Auto Naming:*** An index is added to a base name. When a new base name is set (or the naming scheme changes) a start index is determined automatically by searching the data target directory for files starting with the base-name. Initially (i.e. when ***DET.FRAM.SEQIDX*** is set to zero) the returned index is the highest existing index plus one. If ***DET.FRAM.SEQIDX*** is larger than zero the returned index is the first not existing index which is larger than ***DET.FRAM.SEQIDX***. Once the index is determined it is incremented by one (without further check) after each exposure until either the base-name or the naming scheme changes or a new (minimum-)sequence number is explicitly set via a setup command. This makes it necessary that if ***DET.FRAM.SEQIDX*** is set to a value larger than zero, then no file with the current base-name and an index larger than ***DET.FRAM.SEQIDX*** must exist in the data target directory.

The frame name and frame number are only added to the filename in case individual files are generated for each frame ("***single***" file format).

4.7 Detector Mosaics

Detector mosaics can be handled in various ways. When being all of the same type and when fitting into the computing resources of a single NGC workstation, just the chip dimension (***DET.CHIP.NX/NY***) in the detector configuration file may be enlarged and the chips can be mapped into a single image (virtual chip) with an appropriate sorting map. It is possible to split them up again into image extensions when storing the data to disk and use the full image only for the real-time display. The ***DET.CHIP.SPLITX/Y*** keywords define how many chips are mapped in the full image in x- and y-direction.

Otherwise the ***DET.CHIPS*** keyword can be set to <N> to give the number of chips (or also virtual chips in the above sense). This will automatically set the *-ndet* option of the acquisition process to a value of N / <*number of launched acquisition processes*>. Each acquisition process will thereby be instructed to produce a [*NX x (NY x ndet)]* data frame containing the *ndet* images in consecutive order. The associated acquisition module in the control server would split the frame again into *ndet* separate images and would store them either to individual files or to binary image extensions of a single file. The chip parameters (position in the mosaic, “chip alive”, etc.) as given in the detector configuration file (see section 3.5.2) are stored in the FITS-(extension) headers. If multiple files are generated, a DET<n> extension will be added to the filename:

```
<name>_DET<n>[_<frame-name>][_<frame-number>].fits
```

The index <*n*> gives the index of the first detector stored in this file.

4.8 Post-Processing

Usually post-processing is not in the scope of the detector control software and should be done by higher level SW (data pipeline), especially when FITS information from other sub-systems (TCS, ICS) is needed for proper evaluation. Nevertheless it might be at least convenient to apply some operations before storing the data to disk. Such operations might be image rotation or the evaluation of some data in order to put the result into the FITS-header. As this has a fully application specific nature, it is intended to be done in a post-processing callback, which is executed every time before a frame is stored to disk. If the call-back procedure generates the data file by itself it has to inform the server, that no more operation should be done with this frame. This information can be passed in the return value (*ngcbSKIP*) of the call-back procedure. Some applications also require that this post-processing is done within an infinite loop (i.e. outside the data taking phase during an exposure). A typical example for this are slow control loops with low real-time requirements like secondary autoguiding. The sustained transfer (plus post-processing) is enabled / disabled via a transfer-enable flag (***DET.ACQi.TRANSFER T/F***).

4.9 Exposure Control

Exposures are started using the ***START*** command. The server will perform a snapshot of all relevant parameters to be added to the FITS header(s) of the produced data file(s). Normally, when an exposure is started both sequencer and acquisition are restarted. It is also possible to let the sequencer run continuously, when a ***START***-command is issued. This is controlled via the ***SETUP*** keyword “***DET.SEQi.CONT T|F***”. A default value for this can be given in the detector configuration file. In continuous mode just the acquisition process resets its buffers and counters and starts building a new sum. The counter reset is required to avoid that corrupted data is used for computing the result frames, like if for instance the telescope was moved and frames where taken during the movement. As the exposure start command is sent asynchronously in this case, the current integration needs to be skipped. In the worst case this introduces an overhead equal to the detector integration time. To avoid this overhead the acquisition module can also be set to a “*continuous mode*” with the ***SETUP*** keyword "***DET.ACQi.CONT T|F***". In continuous mode the counter-reset is disabled and it is up to the initiator of the exposure start command to ensure, that there was no change in the field of view since the last integration start. A timed exposure start can be done using the command

```
START -at <hh.mm.ss[.uuuu]>
```

which defines an absolute start time (UTC). Until the actual start time is reached, the exposure status is set to "***pending***", which will limit the set of accepted commands during that time.

It is possible to synchronize exposures on multiple NGCIRSW instances via the external trigger input of the sequencer hardware module (see [AD9]).

If several sequencers are installed in the same system (i.e. the same instance of NGCIRSW), then the exposures can be synchronized by using the global run-signal, which is raised by one sequencer instance and is propagated to all other sequencer instances having the external run-control enabled (***DET.SEQi.RUNCTRL = T*** in the detector configuration file). If one of the sequencers is not operating in continuous mode (i.e. it is stopped and restarted as exposure start), then the ***DET.SEQi.CONT*** keyword is ignored for all other synchronous sequencers.

When the exposure has started, the exposure status will be "***integrating***". When the header of the last file has been received by the data transfer task (i.e. the break-conditions for all frames have been reached) the exposure status goes to "***transferring***". At this time all detector-data for the current exposure are taken and it would be possible to change the field of view (e.g. move the telescope). When the last file has been stored on disk, the exposure state goes to "***success***". If an error occurred during the exposure, the status goes to "***failure***". If the exposure was aborted, the status goes to "***aborted***". The exposure status is stored in the database attribute '***<alias>ngcircon:exposure.expStatus***'. It can take the following values:

```
1   -  INACTIVE
2   -  PENDING
4   -  INTERGRATING
64  -  TRANSFERRING
128 -  SUCCESS (completed_
256 -  FAILURE (completed)
512 -  ABORTED (completed)
```

An explicit status value (in ASCII string format) is stored in the database attribute '***<alias>ngcircon:exposure.expStatusName***'.

Whenever a new data file is created, the full path name is written to the database attribute '***<alias>ngcircon:exposure.newDataFileName***'.

The exposure can be aborted using the ***ABORT***-command. In this case no data file is generated unless a frame was already received on the WS at the time when the command was issued.

The ***END***-command makes the acquisition process terminate the exposure as soon as possible. In this case the generated data file may contain just an intermediate result.

The ***WAIT***-command can be used to wait for an exposure to complete. A reply message with the current exposure state is sent immediately. When the exposure status is (or becomes) "***completed***" (i.e. "***success***", "***failure***" or "***aborted***"), the server sends the last reply, which again contains the actual exposure state.

4.10 Timing Accuracy

The control server will provide timestamps for exposure start and for all received frames with an accuracy of - at least - 0.1 seconds. The accuracy depends mainly on the operating system gitters. If a high accuracy is needed the detector readout can be triggered via the VLT TIM.

4.11 Chopping Mode

Synchronization of detector read-out with a chopper is done via the external trigger input of the sequencer (see [AD9]). If no chopper signal is available for this purpose, then the synchronization is done via the VLT TIM, which has to start a pulse generation at same start time and with same frequency as the chopper (see Figure 4). The chopping frequency has to be rounded to two digits in this case. The NGC hardware will take care, that the sequence always starts in the same phase (e.g. always "*on object*"). A maximum value for the chopping frequency is computed within the sequencer program and is stored in the setup parameter ***DET.CHOP.FREQ (Hz)***. This can either be retrieved via the ***STATUS***-command or also via the database entry '***<alias>ngcircon:chopper.freq***'. Chopping mode is enabled/disabled by the setup parameter ***DET.CHOP.ST (T/F)***. The chopper transition time is passed to the sequencer program via the setup parameter ***DET.CHOP.TRANSTIM (seconds)***. The ***DET.CHOP.ST*** parameter is also passed to the respective acquisition process, which will then take care of computing the subtracted images.

The computation of the result image (*object* - *sky*) is application specific. Usually the parameters ***DET.CHOP.NCYCLES*** and ***DET.CHOP.CYCSKIP*** define the number of chopping cycles and the number of cycles to skip after start, and ***DET.IR.NDIT*** + ***DET.IR.NDITSKIP*** integrations will be done on each chopping half cycle. The ***DET.IR.NDITSKIP*** integrations are skipped at the beginning of each half cycle. Optionally the data for the chopping half cycles can also be computed (controlled via the ***FRAME***-command - see section 4.2). However - these parameters are application specific and may not be valid in all cases. If applicable they will be available either via the ***STATUS***-command or via the parameter table in the database attribute '***<alias>ngcircon:system.param***'.

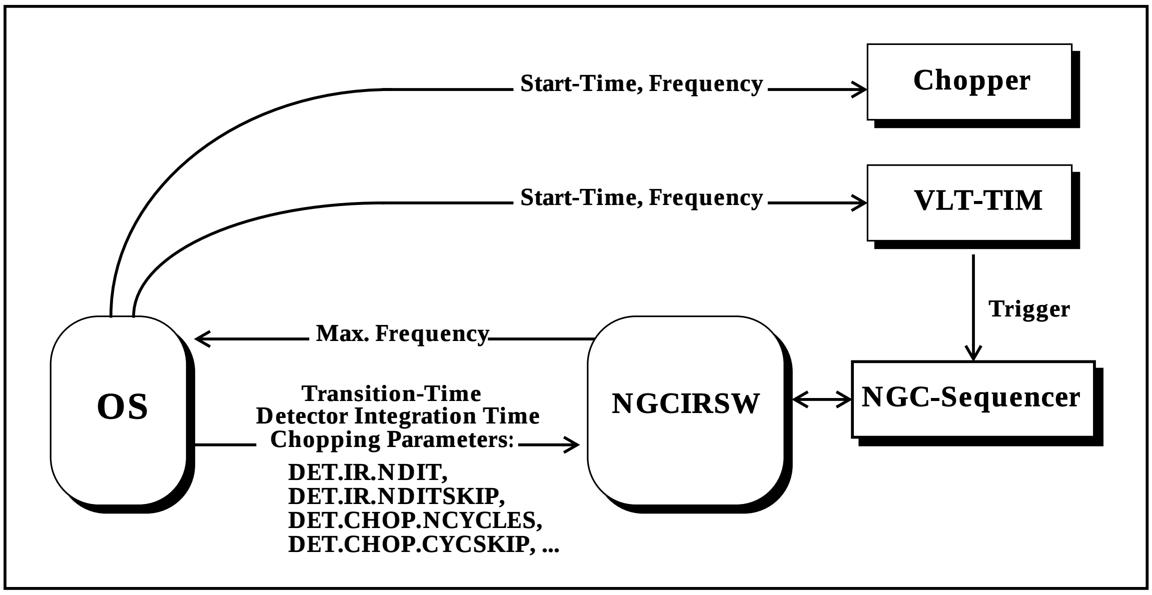


Figure 4 Chopping Mode

5 GRAPHICAL USER INTERFACE

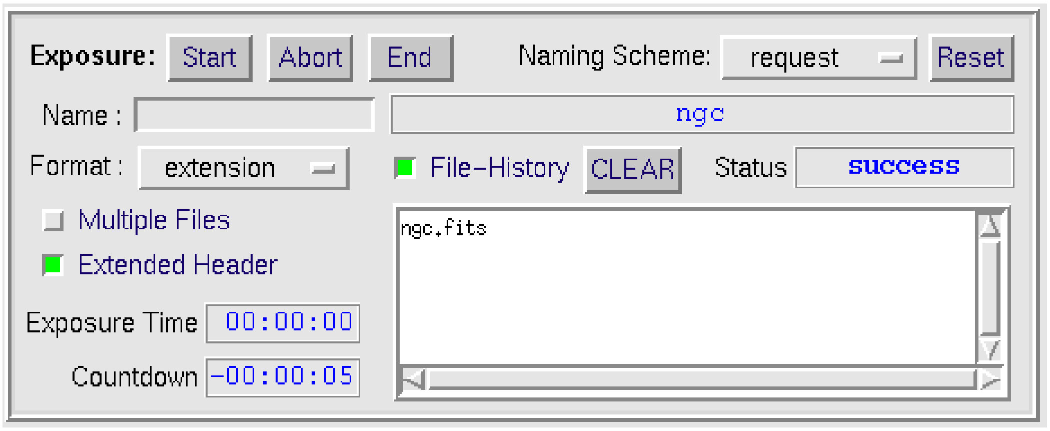
The graphical user interface (GUI) is created with the VLTSW panel editor [RD39]. Commands are sent from the GUI to the control server via the CCS message system. All status data is read back via the database. The panel reacts automatically on database events (i.e. when an attribute in the database changes). Several panels can run in parallel to control different instances of the control server. An instance number can be passed to the GUI via the "*-inst*" command line option. The database branch and the server name, with which the control server is registered in the environment, is derived from this number. A startup script can be used to start both the GUI and the server with the proper instance number.

A new system configuration and a new detector configuration can be loaded via the panel menu bar. There one can also switch between *normal mode* and *simulation mode* and issue all commands for server state switching (see 3.2). It is possible to restart the control server from the GUI. The read-out modes, as defined in the detector configuration, can be selected via a global option button.

5.1 User Interface Classes

The hardware modules (sequencer, CLDC, ADC) and the acquisition modules are represented as instances of UIF classes. There is one UIF class instance per hardware module. The instances can be selected with option buttons outside the class widget. Some commands refer to "all" instances ("*start all sequencers*", "*enable all CLDC modules*") and require a button outside the class widgets.

The panel contains a UIF class for the overall exposure setup, where one can select the naming scheme and the file format, set the exposure filename and issue all exposure control commands by pressing buttons (***START***, ***ABORT***, ***END***). The produced frames are displayed in a file history widget. It is possible to show also the FITS header of the produced FITS-files by double clicking on the file names.



Exposure: Start Abort End

Naming Scheme: request

Reset

Name :

ngc

Format : extension

🟩 File-History CLEAR

Status success

☐ Multiple Files

🟩 Extended Header

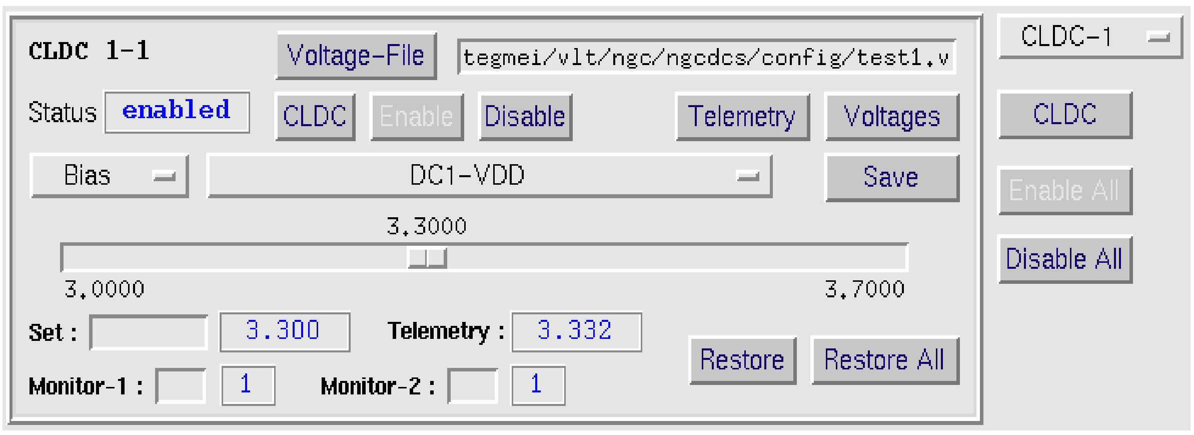
Exposure Time 00:00:00

Countdown -00:00:05

```
ngc.fits
```

Figure 5 Exposure UIF Class

The CLDC UIF class provides a voltage tune bar to tune a selected clock- or bias-voltage within the defined range. The ranges are read from the database. The actual voltages can be saved to a voltage configuration file. New voltage configuration files can be loaded by typing the file name into an entry-widget or via a file browser. Voltage telemetry is shown in a global result widget when pressing the telemetry button.



CLDC 1-1

tegmei/vlt/ngc/ngcdcs/config/test1.v

Status enabled

Bias

DC1-VDD

3.3000

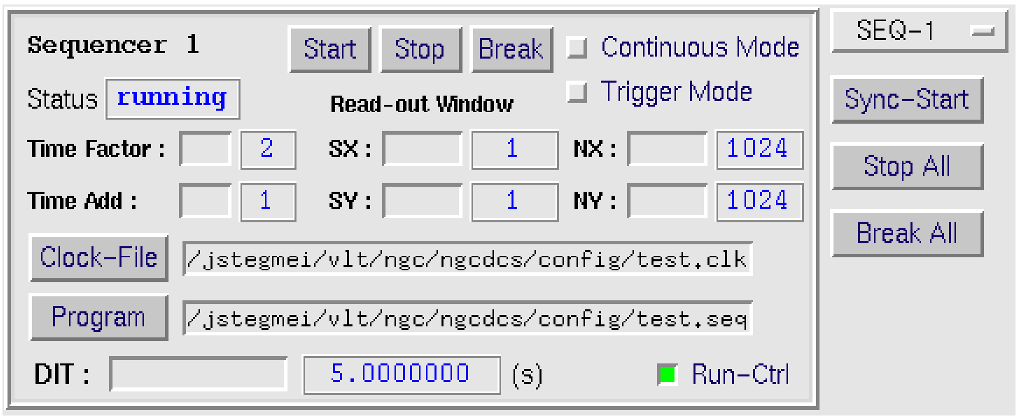
3.0000 3.7000

Set : 3.300 Telemetry : 3.332

Monitor-1 : 1 Monitor-2 : 1

Figure 6 CLDC UIF Class

The sequencer UIF class provides all means to setup and control one sequencer instance. Clock pattern configuration files and sequencer programs can be loaded by typing the file name into entry-widgets or via a file browser. The class also contains the entry widget for the detector integration time, which can be set individually for each sequencer in the system.



Sequencer 1

Status running

Read-out Window

Time Factor : 2 SX : 1 NX : 1024

Time Add : 1 SY : 1 NY : 1024

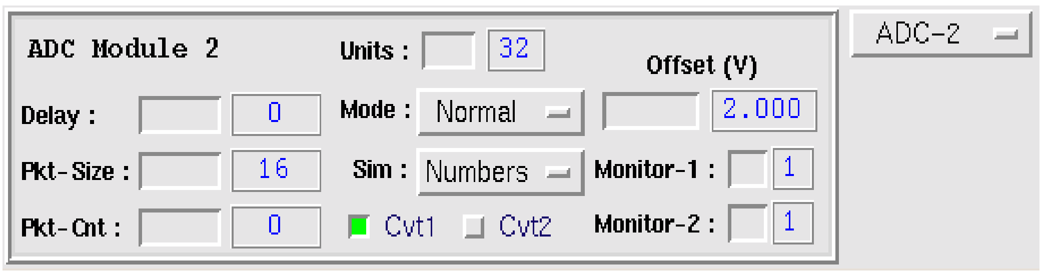
/jstegmei/vlt/ngc/ngcdcs/config/test.clk

/jstegmei/vlt/ngc/ngcdcs/config/test.seq

DIT : 5.0000000 (s) 🟩 Run-Ctrl

Figure 7 Sequencer UIF Class

The ADC UIF class provides all means to setup and control one instance of an ADC module:



ADC Module 2

Units : 32

Offset (V)

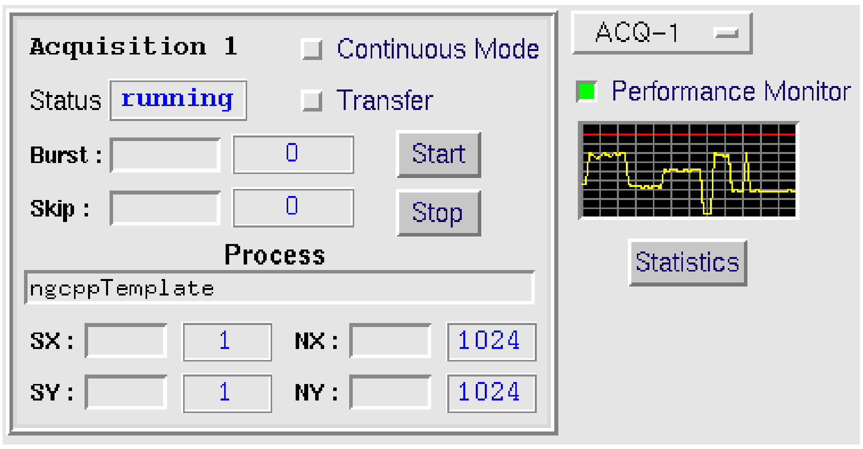
Delay : 0 Mode : Normal 2.000

Pkt-Size : 16 Sim : Numbers Monitor-1 : 1

Pkt-Cnt : 0 🟩 Cvt1 ☐ Cvt2 Monitor-2 : 1

Figure 8 ADC UIF Class

Via the acquisition UIF class a new acquisition process can be launched (usually this will happen automatically when selecting the read-out mode):



Status: running
Burst : 0
Skip : 0
Process: ngcppTemplate
SX : 1
NX : 1024
SY : 1
NY : 1024

Figure 9 Acquisition Process UIF Class

The window for the acquisition process is usually taken over from the sequencer module class (read-out window). But it is also possible to set it manually (mainly used for test purposes).

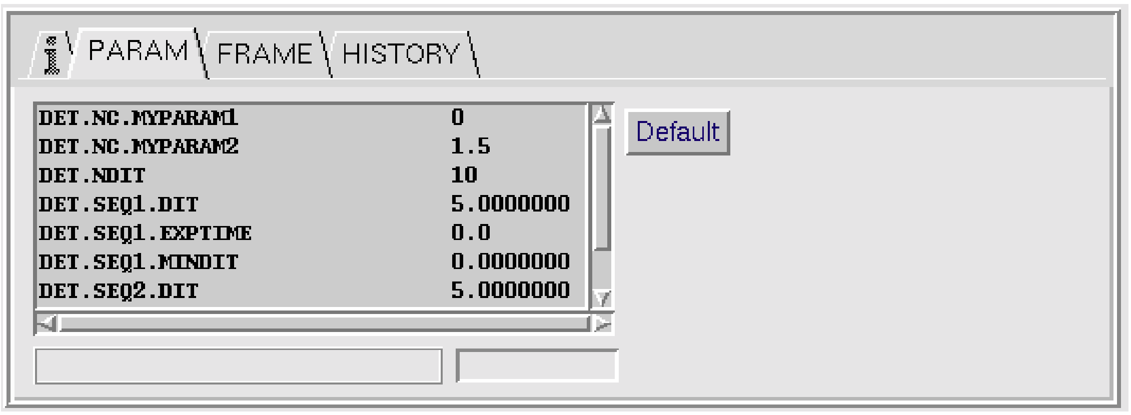
The performance monitor in the acquisition UIF Class shows the pre-processing CPU load on the NGC-LCU, where the current process is launched.

All of the hardware- and acquisition-module classes show the actual status ("*sequencer running*", "*CLDC enabled*", "*acquisition process running*", "*ADC enabled*").

5.2 Notebook Area

There is a notebook area where one can select additional UIF classes. By default there is one for the parameter setup, one for the frame setup and one for the action history. It is possible to add application specific control widgets in that area (chopper interface, etc.).

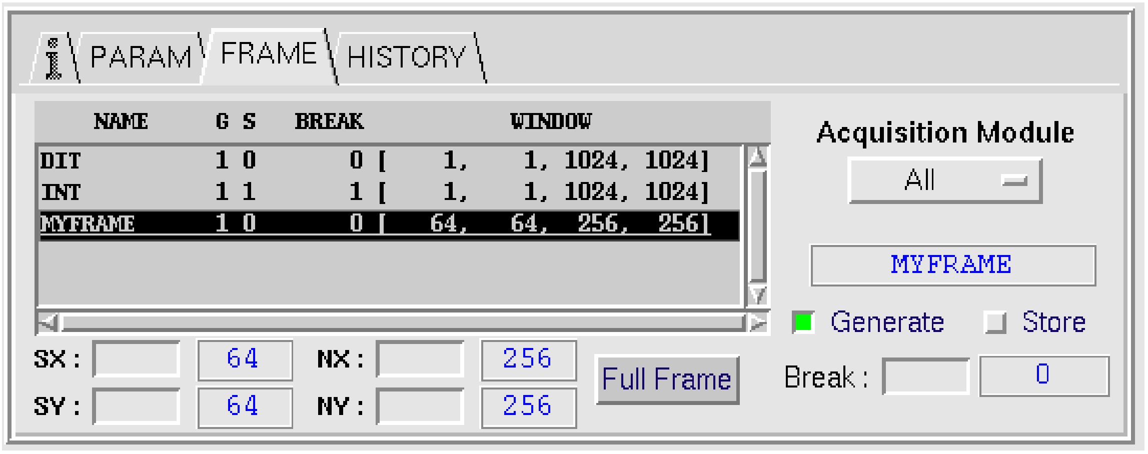
The parameter setup widget shows the list of application specific parameters as described in section 3.5.4. There is a "***Default***" button, which will set all parameters to the values given in the parameter default setup files. The default setup files are applied in the order: system configuration default setup, detector configuration default setup, read-out mode default setup, where the last loaded setup has precedence.



```
DET.NC.MYPARAM1        0
DET.NC.MYPARAM2        1.5
DET.NDIT               10
DET.SEQ1.DIT           5.0000000
DET.SEQ1.EXPTIME       0.0
DET.SEQ1.MINDIT        0.0000000
DET.SEQ2.DIT           5.0000000
```

Figure 10 Parameter Notebook

In the frame notebook one can select the frames to be generated and the ones to be stored during the next exposure. The exposure break condition and a software window can be set for each defined frame type. The frame setup can be done either individually per acquisition module or globally for all acquisition modules.



```
NAME          G S    BREAK          WINDOW
DIT           1 0        0 [    1,     1, 1024, 1024]
INT           1 1        1 [    1,     1, 1024, 1024]
MYFRAME       1 0        0 [   64,    64,  256,  256]
```

MYFRAME

🟩 Generate ☐ Store

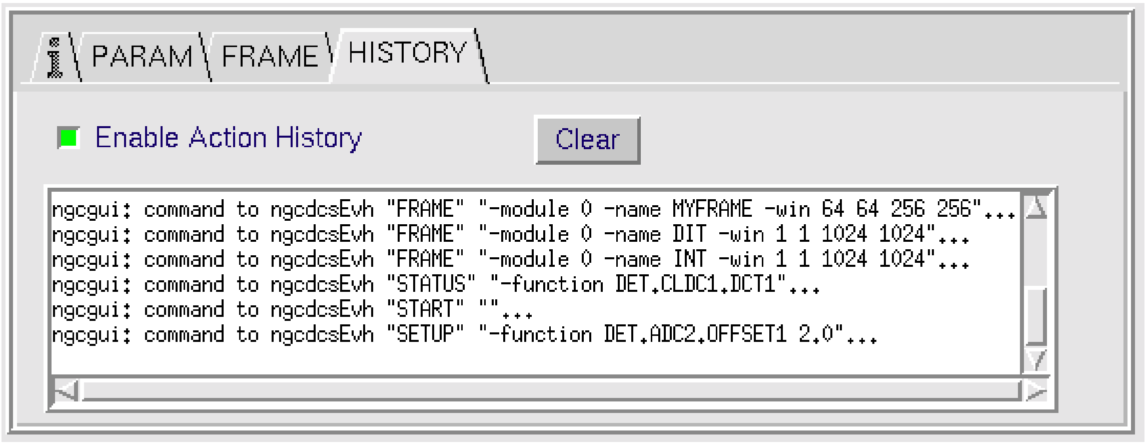
SX : 64 NX : 256

SY : 64 NY : 256

Break : 0

Figure 11 Frame Setup Notebook

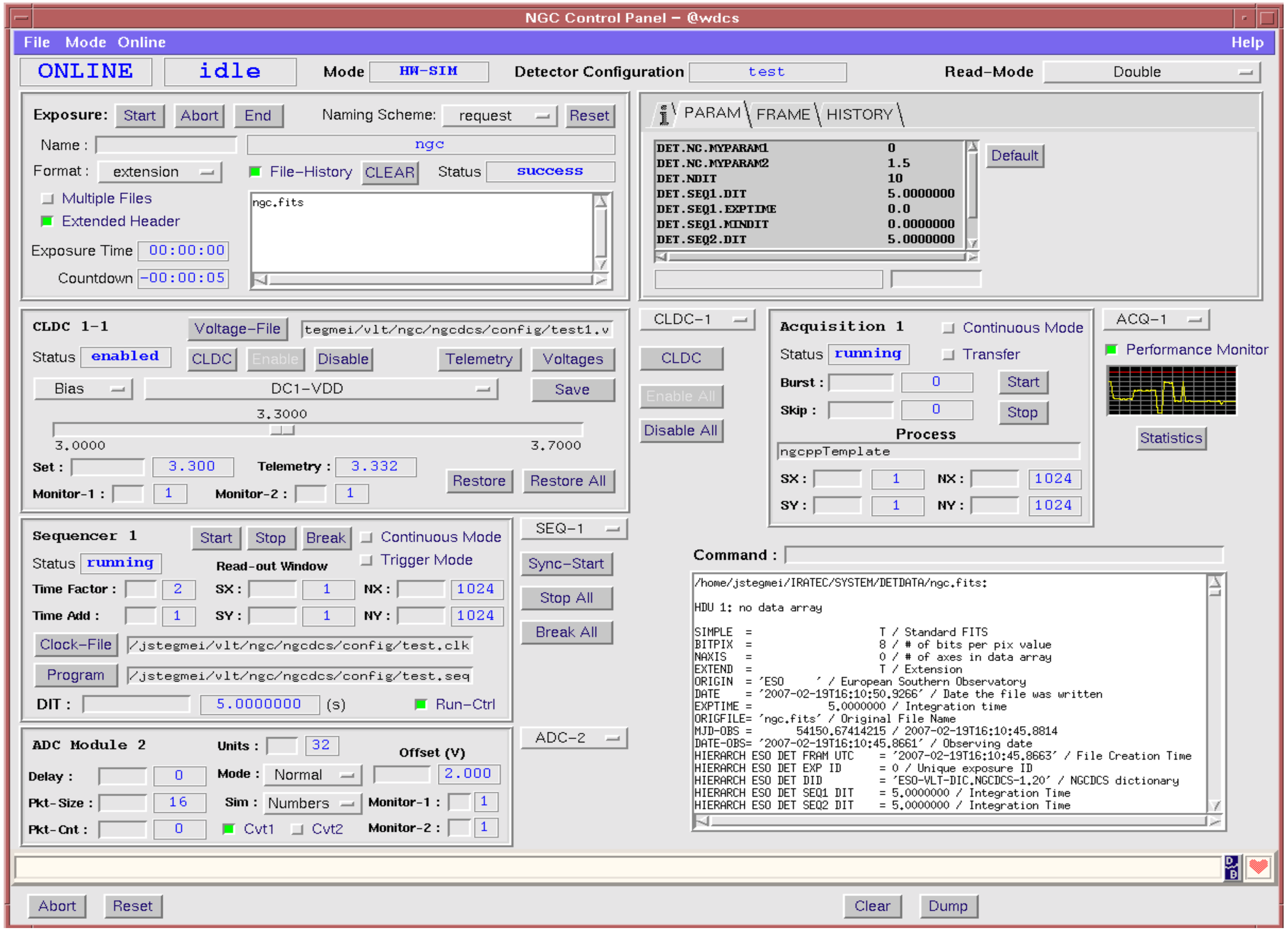
The history notebook logs all major actions (server restart etc.) which are done via the panel.



🟩 Enable Action History

```
ngcgui: command to ngcdcsEvh "FRAME" "-module 0 -name MYFRAME -win 64 64 256 256"...
ngcgui: command to ngcdcsEvh "FRAME" "-module 0 -name DIT -win 1 1 1024 1024"...
ngcgui: command to ngcdcsEvh "FRAME" "-module 0 -name INT -win 1 1 1024 1024"...
ngcgui: command to ngcdcsEvh "STATUS" "-function DET.CLDC1.DCT1"...
ngcgui: command to ngcdcsEvh "START" ""...
ngcgui: command to ngcdcsEvh "SETUP" "-function DET.ADC2.OFFSET1 2.0"...
```

Figure 12 Action History Notebook



ONLINE idle Mode HW-SIM Detector Configuration test Read-Mode Double

Exposure: Naming Scheme: request
Name : ngc
Format : extension File-History Status success
Multiple Files
Extended Header
Exposure Time 00:00:00
Countdown -00:00:05
ngc.fits

PARAM FRAME HISTORY

```
DET.NC.MYPARAM1          0
DET.NC.MYPARAM2          1.5
DET.NDIT                 10
DET.SEQ1.DIT             5.0000000
DET.SEQ1.EXPTIME         0.0
DET.SEQ1.MINDIT          0.0000000
DET.SEQ2.DIT             5.0000000
```

CLDC 1-1 tegmei/vlt/ngc/ngcdcs/config/test1.v
Status enabled
Bias DC1-VDD
3.3000
3.0000 3.7000
Set : 3.300 Telemetry : 3.332
Monitor-1 : 1 Monitor-2 : 1

CLDC-1

Acquisition 1 Continuous Mode
Status running Transfer
Burst : 0
Skip : 0
Process
ngcppTemplate
SX : 1 NX : 1024
SY : 1 NY : 1024

ACQ-1
Performance Monitor

Sequencer 1 Continuous Mode
Status running Read-out Window Trigger Mode
Time Factor : 2 SX : 1 NX : 1024
Time Add : 1 SY : 1 NY : 1024
/jstegmei/vlt/ngc/ngcdcs/config/test.clk
/jstegmei/vlt/ngc/ngcdcs/config/test.seq
DIT : 5.0000000 (s) Run-Ctrl

SEQ-1

ADC Module 2 Units : 32 Offset (V)
Delay : 0 Mode : Normal 2.000
Pkt-Size : 16 Sim : Numbers Monitor-1 : 1
Pkt-Cnt : 0 Cvt1 Cvt2 Monitor-2 : 1

ADC-2

Command :

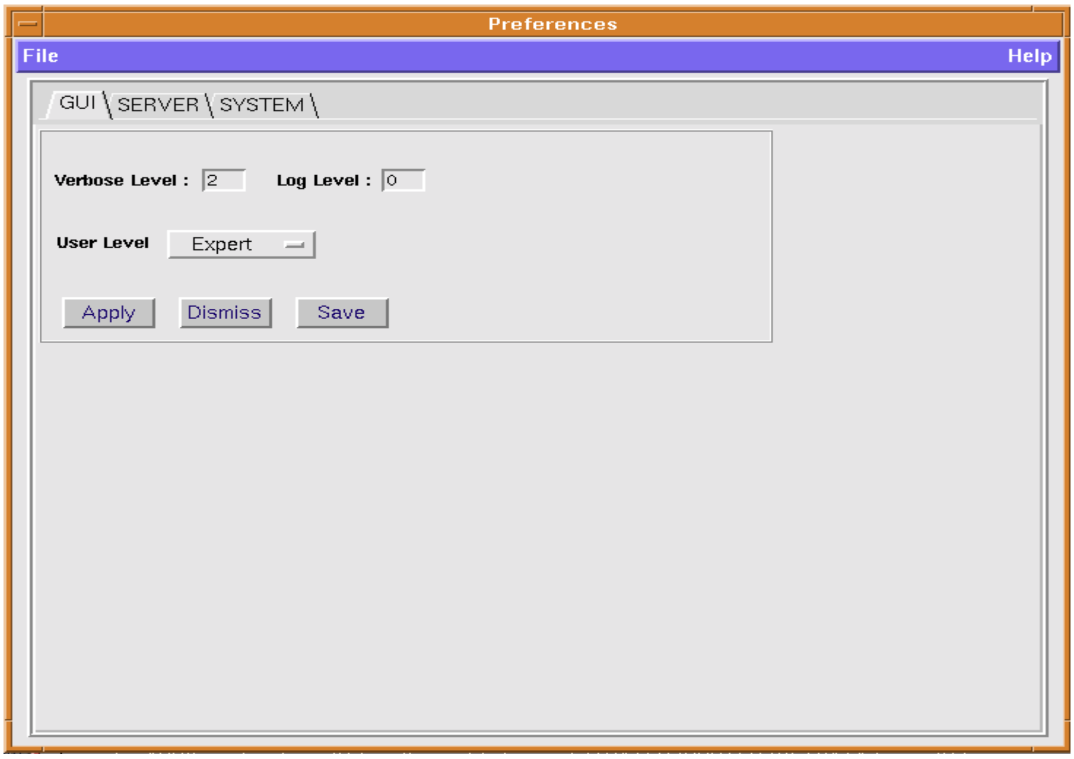
```
/home/jstegmei/IRATEC/SYSTEM/DETDATA/ngc.fits:

HDU 1: no data array

SIMPLE  =                    T / Standard FITS
BITPIX  =                    8 / # of bits per pix value
NAXIS   =                    0 / # of axes in data array
EXTEND  =                    T / Extension
ORIGIN  = 'ESO     ' / European Southern Observatory
DATE    = '2007-02-19T16:10:50.9266' / Date the file was written
EXPTIME =            5.0000000 / Integration time
ORIGFILE= 'ngc.fits' / Original File Name
MJD-OBS =       54150.67414215 / 2007-02-19T16:10:45.8814
DATE-OBS= '2007-02-19T16:10:45.8661' / Observing date
HIERARCH ESO DET FRAM UTC    = '2007-02-19T16:10:45.8663' / File Creation Time
HIERARCH ESO DET EXP ID      = 0 / Unique exposure ID
HIERARCH ESO DET DID         = 'ESO-VLT-DIC.NGCDCS-1.20' / NGCDCS dictionary
HIERARCH ESO DET SEQ1 DIT    = 5.0000000 / Integration Time
HIERARCH ESO DET SEQ2 DIT    = 5.0000000 / Integration Time
```

Figure 13 Engineering GUI

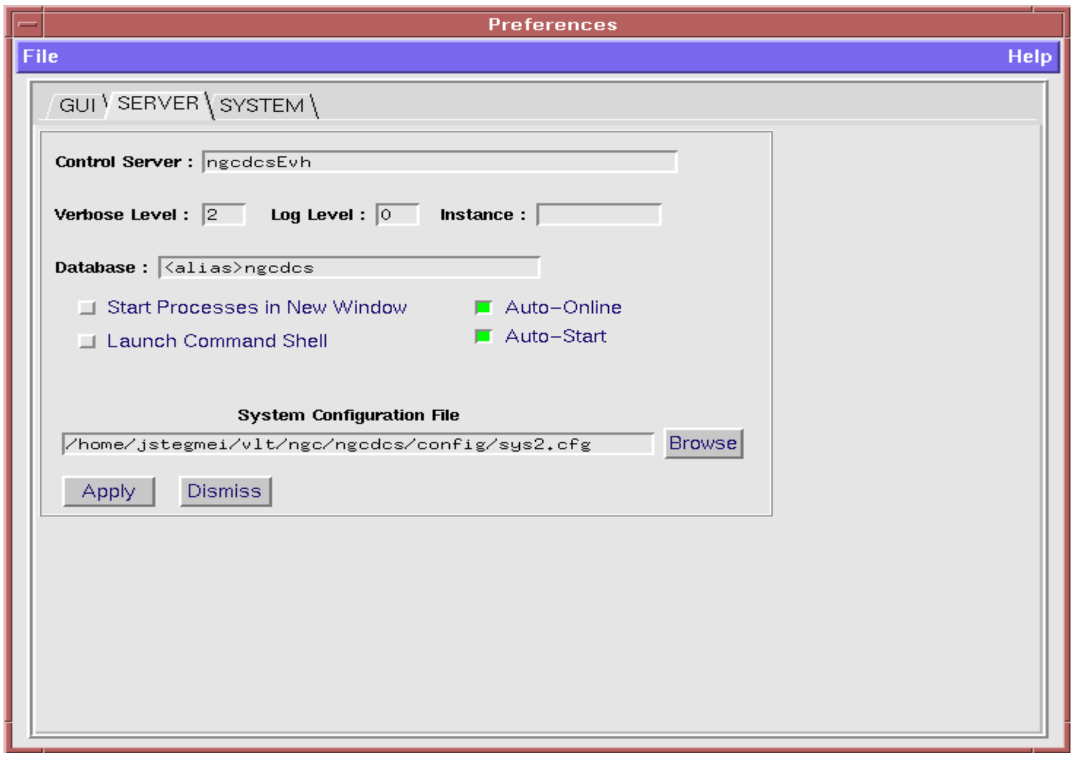
5.3 Server Preferences and System Configuration



Verbose Level : 2 Log Level : 0

User Level Expert

Figure 14 GUI Preferences



Control Server : ngcdcsEvh

Verbose Level : 2 Log Level : 0 Instance :

Database : <alias>ngcdcs

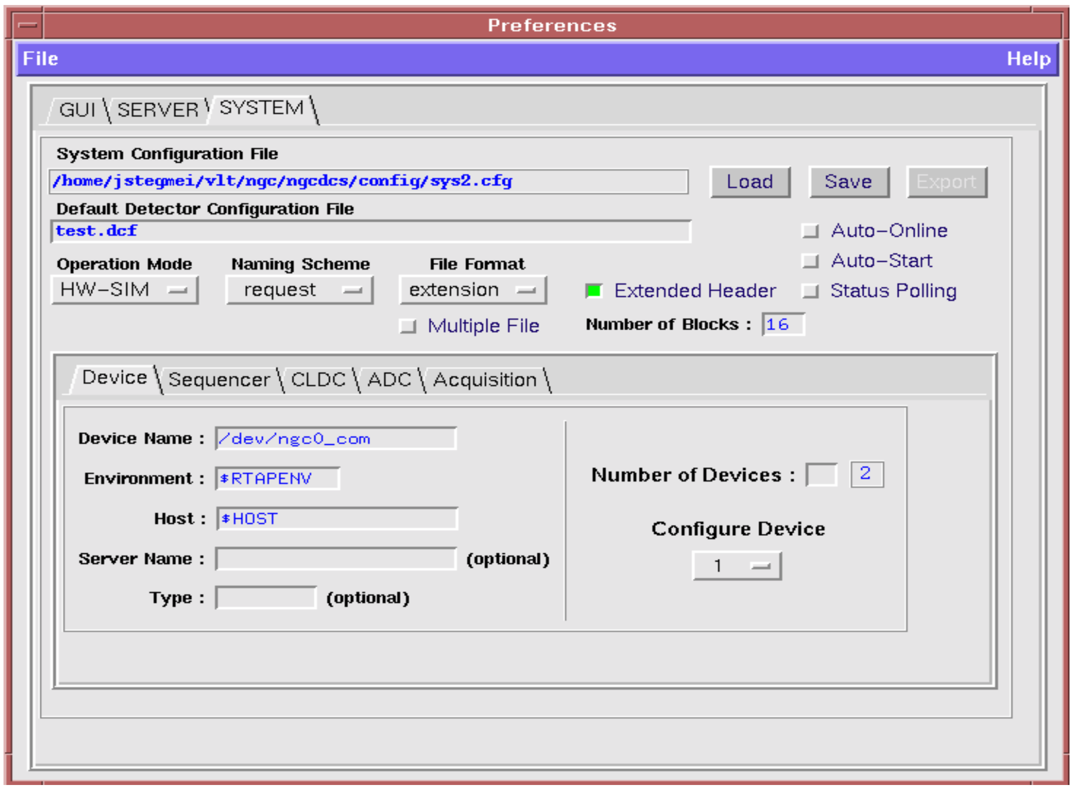
☐ Start Processes in New Window ■ Auto−Online

☐ Launch Command Shell ■ Auto−Start

System Configuration File

/home/jstegmei/vlt/ngc/ngcdcs/config/sys2.cfg

Figure 15 Server Preferences



System Configuration File: /home/jstegmei/vlt/ngc/ngcdcs/config/sys2.cfg

Default Detector Configuration File: test.dcf

Operation Mode: HW-SIM

Naming Scheme: request

File Format: extension

Multiple File ☐

Extended Header 🟩

Number of Blocks : 16

Auto-Online ☐

Auto-Start ☐

Status Polling ☐

Device Name : /dev/ngc0_com

Environment : $RTAPENV

Host : $HOST

Server Name : (optional)

Type : (optional)

Number of Devices : 2

Configure Device: 1

Figure 16 System Configuration Tool

6 TRACEABILITY MATRIX

6.1 NGC Requirements from [AD6]

Item	Requirement	Section
	3.7 Software	
1	Generally, software shall not limit the performance of the hardware.	2.1
2	It shall be command driven.	2.1, 2.2
	3.7.1 High-level operating systems	
3	The high-level operating systems must be |compliant with the VLT requirements. However, their number and diversity shall be kept to the minimum necessary for NGC.	2.1
4	A careful attempt shall be made to define an interface layer between the NGC control software proper and the operating system(s) and so to enable porting of all software above this layer at reasonable cost.	[AD9]
	3.7.2 Configuration Control	
5	At all times, all software and all parameter files shall be kept under configuration control.	2.3, 3.5.4
6	For critical parameter files, an additional mechanism to ensure their integrity (e.g., check sums) should be considered	TBD
	3.7.3 Programming	
7	The usage of modern code-generating tools with a view towards testing, documenting, and debugging is encouraged. Their selection should be coordinated with the Technical Division. Island solutions should be avoided.	TBD

Item	Requirement	Section
8	For each module, code and documentation shall be designed such that it can be maintained without analyzing other modules.	-
	3.7.4 Installation and start-up procedures	
9	Fully automatic installation procedures and versatile configuration tools shall be provided.	2.3, 3.5.1, 3.5.2
10	Execution of the standard control software in the telescope environment shall not require any special user privileges to be granted by the operating system.	2.2
11	The start-up script shall not require more than 10 s for auto-recognition of the hardware and the ready-for-use initialization of hard- and software.	No further overhead here.
	3.7.5 Resource checking	
12	Software shall be able, prior to each exposure, to check the availability of all critical resources.	4.6
	3.7.6 Elementary functions	
13	The set of elementary functions shall comprise those of IRACE and FIERA.	7.2.2
14	The addition of further functions shall be possible without affecting the others.	3.8, [AD9]
	3.7.7 Tests	
15	Test software shall be developed in parallel to the control software itself.	2.4, [AD9]
16	The emulation of failures of other utilities (software, hardware, network, lack of resources, access denial) should be considered.	3.4, 3.6, [AD9]

Item	Requirement	Section
17	Standardized tests of the software corresponding to any supported hardware configuration shall be possible by merely selecting a single set of parameters.	3
18	A sequence of tests of several hardware configurations shall be possible without operator intervention.	3
19	Means should be considered to let NGC keep track of the frequency of usage of its key functionalities as a way to set usage-oriented test priorities.	TBD
	3.7.8 Times and timings	
20	Without the VLT TIM, all absolute times shall be correct to within less than 0.1s.	4.9
21	Relative synchronizations and time intervals shall be accurate to better than 0.1% or, for intervals less than 10s, to better than 0.01s.	-
22	Stricter timing requirements shall be realized using TIM.	4.9
	3.7.9 Special modes	
23	Support of the following techniques (in the order of decreasing priority) should be foreseen: • nod and shuffle • subpixel sampling and digital filtering so that during an exposure the built-up of the S/N can be followed by performing a regression analysis for each pixel • drift scanning • non-destructive readout • on-chip charge shifts by a user-definable amount (e.g., for through-focus sequences)	[AD9]
24	Device type-specific modes offered by state-of-the-art IR detectors shall be included.	[AD9]

Item	Requirement	Section
25	If centroiding functions need to be supported, this shall be possible at a frame rate of 1 Hz for data arrays of up to 256 x 256 pixels using a single Gaussian fit or similar. For much smaller data arrays, rates of up to 100 Hz should be possible.	[AD9]
	3.7.10 Consecutive exposures	
26	If, e.g. due to on-line data processing, the time between end of detector readout and availability of the FITS file on disk becomes a significant overhead, it shall be possible to configure the software such that the next exposure begins right after the previous readout.	4.9
	3.7.11 Windowing and on-chip binning	
27	Standard windowing and on-chip binning shall be provided	4.2, 4.3
28	The number of windows should only be limited by the capabilities of the detectors.	4.2, 4.3
	3.7.12 Pixel processor	
29	A pixel processor shall be embedded in the system. Its interfaces to the remainder of the system shall be designed such that a replacement of the hardware plus operating system and/or of the processing software can be fully transparent to all other subsystems.	2.1, 2.2, [AD9]
30	The following operations shall be supported from the beginning: averaging of frames with and without removal of outliers (e.g., particle events) • bias subtraction • centroiding of point sources • TBC (If performance reasons so require, the implementation may be detector dependent.)	[AD9]
31	Close integration with NGC of a general-purpose image processing system featuring a user friendly scripting language could be considered.	[AD9]

Item	Requirement	Section
32	More desirable is an interface to the ESO DFS and the inclusion of general-purpose algorithms and recipes in the DFS CPL for re-use by data reduction pipelines.	2.1
	3.7.13 ALMA control software	
33	If this is in the general interest of ESO and supported by the Technology Division, elements of the ALMA control software may be used.	TBD
	3.7.14 Special utilities	
34	For multi-port systems, bias equalization to within better than 1% shall be possible on demand but without any further operator supervision.	Not for IR.
	3.8 External interfaces	
35	Ideally, external interfaces (e.g., commands, databases) presently maintained by IRACE and FIERA would be supported by NGC with a minimum of changes so as to make the integration of NGC with the ESO operations scheme as seamless as possible. However, since in this regard the commonalities of FIERA and IRACE are very limited, this also limits backward compatibility. In no case shall NGC feature two different types of interfaces for the same purpose.	TBD
	3.8.1 Data format	
36	The data format shall be compliant with the Data Interface Control Document.	4.5
37	Comprehensive detector and electronics telemetry shall be included in the data headers.	4.5, 4.9
38	From the FITS headers, it shall be possible to uniquely infer the complete set of hard- and software configuration and all parameter values.	4.9

Item	Requirement	Section
39	A generalized, moderately configurable interface to real-time computers, e.g. for adaptive optics or fringe tracking applications, shall be defined (in cooperation with ESO software engineers working downstream from such an interface).	Not within this scope.
40	It could be advantageous that (a possibly special incarnation of) the pixel processor can serve as the real-time computer (or vice versa).	TBD
41	Latency shall not exceed 100 us.	Not within this scope.
	3.8.3 Real-time display	
42	An interface to the RTD shall be provided.	2.1
43	For high frame rates, it shall be possible to request only every nth frame to be displayed.	[AD9]
44	Adaptive auto-selection shall be supported.	[AD9]
	3.8.4 VLT telescope control system	
45	It shall be possible to synchronize detector operations with the following functions: • Telescope nodding • M2 chopping • Non-sidereal tracking	[AD9]
	3.8.5 VLT time distribution system	
46	The possibility of an interface to the VLT Time Interface Module shall be foreseen.	[AD9]
47 - 52	Reserved.	
	3.11 Diagnostic tools	

Item	Requirement	Section
	3.11.1 Hardware self-test	
53	The hardware shall be able to execute a comprehensive self-test. It shall be possible to start it by pressing a physical button as well as by software. Due consideration shall be given to the protection of the detectors. The execution shall not exceed 5 minutes. An automatic log shall be produced.	TBD
54	In order to save space on the electronics boards, it is acceptable to let remote software (e.g., on the xLCU) execute these tests with hardware only reporting its status. This software shall be developed in parallel to the one of the hardware.	TBD
	3.11.2 Read-back of parameter values	
55	It shall be possible to read back the actual values of all parameters set by software.	3.5.4
	3.11.3 Automatic identification of hardware components	
56	All LRUs (line Replaceable Unit) shall have a unique identification that is readable by software.	3.2, [AD9]
57	An extension also to detectors shall be considered.	TBD
58	Software shall be able to use this information for auto-configuration.	3.2, 3.5.1, [AD9]
	3.11.4 Error handling	
59	Meaningful error messages and log files are essential; they shall enable software staff not familiar with the software or its scope to identify and fix minor problems. Different severity levels shall be distinguished. The status and options for the next actions shall be clear at all times.	3.3, 3.4
60	It shall be possible to set the severity level up to which automatic recoveries from errors shall be attempted.	TBD

Item	Requirement	Section
61	After a failed data saving, the OS shall have the possibility to recover the last frame.	Not applicable for IR exposures.
62	After an interruption in the power supply, software should be able to automatically restore the status at the beginning of the last successful exposure.	Not possible.
	3.12 Support of engineering work	
	3.12.1 Engineering mode	
63	There shall be a password-protectable engineering mode. It may contain extra modules and options while otherwise may be omitted for reasons of convenience. However, modules used for normal operations shall be identical.	3.9, [AD9]
64	This mode shall offer access to all essential elementary detector control functions and allow hardware engineers rapid proto-typing of experimental software.	3.9, [AD9]
	3.12.2 Change of parameters	
65	A change of software-configurable parameters shall not require a re-start of the system and, where possible, be supported also during readout. This would also benefit multi-mode instruments where, e.g., imaging and spectroscopy require different parameter sets for optimal performance. Switching between modes shall not lead to any hysteresis.	3.5.4, 4.1, [AD9]
66	A mechanism shall be implemented to reduce the risk of out-of-range parameter values being set accidentally that could damage the connected detector(s). One possibility might be to let the controller hardware request a unique electronic ID (such as the serial number) from the detector.	TBD
67	An interface to BOB shall be provided that permits parameter values to be set from dedicated observation blocks / observing templates. To take advantage of this, laboratory setups would need to be able to emulate VLT-compatible instruments to the extent that VLT control software Sequencer scripts can be executed.	2.1

Item	Requirement	Section
	3.12.3 Detector library	
68	A repository with parameter files for specific detector types and their baseline operating modes shall be offered. For engineering purposes, easy copying and editing of such files shall be supported. To the extent possible, different installations of comparable detector systems shall share these data.	3.5.2
	3.12.4 Disabling of components	
69	It shall be possible to declare LRUs and channels defunct. In response to this, the software should be able to automatically adapt itself to the remaining hardware configuration.	3.5.1, 3.5.2, 4.7
	3.12.5 Special modes	
70	The following shall be foreseen: • pocket pumping • convenient connection of monitoring equipment such as oscilloscopes, multimeters, and logic analysers • determination of PTF of DC-coupled IR and CMOS devices by a capacitive comparison technique	[AD9]
	3.12.6 Programming interface	
71	Thought shall be given to the provision of an efficient programmer's interface, ideally with a standard scripting language such as Tcl/Tk, that permits engineers rapid proto-typing of detector control and data processing software.	2.1, [AD9]
	3.12.7 Test facility	
72	A cost-effective test facility for all types of LRUs shall be supplied. It may either be integrated into the controller or stand-alone.	Not within this scope.
73	Its software shall use the one of the NGC only.	Not within this scope.
74	An expandible collection of standard test functions shall be considered.	Not within this scope.

Item	Requirement	Section
75	Where applicable, test results should also be offered in graphical form with an option for hardcopies.	Not within this scope.
	3.12.8 Simulation modes	
76	Standard VLT simulation modes shall be supported. Simulation should be one of the standard modes of NGC rather than an add-on.	3.6, [AD9]
77	To the greatest possible extent, simulation of key elements shall be supported in both soft- and hardware.	3.6, [AD9]
78	Hardware simulators to generate programmable test pixel patterns and video waveforms shall be devised.	Not used.
79	Software simulators shall be hierarchically structured and permit the simulation of data streams with real numbers and realistic data rates so that relative timings, etc. can be tested.	3.6, [AD9]

6.2 NGC Software Requirements from [AD7]

Item	Requirement	Section
	3.1 Functional Requirements	
	3.1.1 Common Requirements	
80	NGCIRSW shall handle at least TBD clocks, TBD biases, TBD preamps and TBD video channels.	3.5.1, 3.5.2, [AD9]
81	NGCIRSW will implement, as a minimum, the commands already used by FIERA and IRACE and described in their CDTs with an interface which will allow backward compatibility.	7.2.2, [AD9]
82	The ONLINE status requires that all voltages are loaded and switches closed as well as telemetry is acquired and checked.	3.2

Item	Requirement	Section
83	NGCIRSW will handle multiple independent detectors.	3.5.1, 3.5.2, 4.7
84	It shall be possible to read number of windows limited only by detector properties	4.3
85	Windows shall be read either in hardware through sequences or in software. The latter case implies that a full frame is read out and then a window of data is computed in memory.	4.2, 4.3
86	Telemetry shall be available at all times, with the possibility to have a separate period for logging on the VLT logMonitor.	3.2
87	NGCIRSW shall transfer all computed results to display (RTD) and/or to FITS-file.	2.1, 4.2, 4.9, [AD9]
88	Display visualization is done in parallel to all other data transfers (i.e. one may look at the DIT frame while storing the INT frame to disk and while the next data is already being processed).	[AD9]
89	If processing- or data-transfer-bandwidth exceeds the capacity of one single computer, the task is split up to N computing units.	2.1, 4.5, [AD9]
90	In order to test the image data path, NGC must be able to produce pre-defined data,	[AD9]
	3.1.3 Infrared Specific Requirements	
91	Each Pixel can be read out N times and an average is computed.	[AD9]
92	Subsampling and digital filtering of individual pixels shall be possible.	[AD9]
93	The ONLINE status requires that the system also starts readout.	[AD9]
94	In case reference values on special channels are read out (e.g. Hawaii2RG), NGC shall be able to interpolate through rows or columns.	7.2
95	Chopping mode shall be implemented	[AD9]

Item	Requirement	Section
96	NGCSW shall be able to transfer bursts of raw data to FITS files.	4.4
97	Standard read-out modes: • Uncorrelated • Double correlated (reset-read-read) • Double correlated (read-reset-read) • Least square fit • Fowler Sampling	[AD9]
	3.2 External Interface Requirements	
	3.21 User Interfaces	
98	NGCIRSW user interfaces will be developed following rules described in [RD39].	5
99	The user interface for telescope operations will merge functionalities of current FIERA and IRACE user interfaces.	TBD
100	Specific graphical interface may be developed in order to ease engineer's work in the laboratory. These interfaces may be developed not following the standards if their use is confined to laboratory.	5
	3.2.3 Software Interfaces	
101	Sequencer programming shall be implemented using a scripting language.	[AD9]
102	The scripting language will allow evaluation of arithmetic formulas at run-time.	[AD9]
103	The startup overhead of the script must be as short as possible.	[AD9]
104	Where no script is required a simple parsing can be done.	[AD9]
105	A graphical tool shall be implemented in order to have also the possibility to program and visualize the sequences.	Not within this scope.

Item	Requirement	Section
106	The format on disk of the sequences shall be ASCII.	[AD9]
107	Other formats, non ASCII, for ancillary data (e.g. for graphics) could be used, but must not be required for running the system.	[AD9]
108	The sequence programming will allow free use of setup parameters.	[AD9]
109	The chopper frequency may be input parameter or output parameter of the sequencer program.	[AD9]
110	Free placement of synchronization points with external trigger.	[AD9]
111	Read-out of multiple windows shall be configurable in programming.	[AD9]
112	Sub-pixel sampling (N samples per pixel) shall also be possible	[AD9]
113	The sequencer programming tool will also allow to emulate the real sequencer.	3.6, [AD9]
114	Interface to standard RTD shall be provided.	2.1, [AD9]
115	The interface and the library to be used are given in [RD40].	[AD9]
116	If needed flat-field frame and bad pixels mask shall be uploaded to NGCIRSW which will then distribute them to the relevant subsystem.	[AD9]
	3.2.4 Communications Interfaces	
117	Communication between internal subsystems of NGCIRSW shall be implemented using VLTSW standard messaging tools as well as CCS database.	2.1, [AD9]
118	Whenever not otherwise specified, the same standards will be used for all other communication interfaces.	[AD9]

Item	Requirement	Section
	3.3.2 Data Transfer	
119	Data transfer rate to the IWS shall be limited only by readout speed.	[AD9]
120	An overhead of max. 5 seconds will be considered acceptable.	[AD9]

6.3 Adaptive Optics Requirements for NGC from [AD20]

Item	Requirement	Section
AONGCREQ-004	Number of detectors controlled from single NGC: 1-4	3.5.1, 3.5.2, [AD9]
AONGCREQ-005	It should be possible to control multiple detectors with a single NGC.	3.5.1, 3.5.2, 4.7, [AD9]
AONGCREQ-006	The detectors controlled by a single NGC all be read with the same read-mode and frame rate. Operations such as start and stop, should operate on all detectors simultaneously.	4.1, [AD9]
AONGCREQ-007	It should be possible to synchronize the start of a read sequence between NGCs	4.9, [AD9]
AONGCREQ-008	The synchronization should allow the start of the readout in separate NGCs to be started within the frame jitter time.	[AD9]
AONGCREQ-009	Ability to visualise 1 of N frames asynchronously on instrument WS using standard RTD, the value of N will be chosen to allow visualisation on the WS of 1-4Hz, the maximum frame rate required to the WS will be 50Hz.	[AD9]
AONGCREQ-010	Ability store 1 of N frames asynchronously in FITS format on instrument WS, the value of N will be defined to allow a maximum frame rate of 50Hz.	[AD9]

Item	Requirement	Section
AONGCREQ-011	Ability to store a series of N guaranteed consecutive frames, where N should be sufficient to store a minimum of 2 seconds of data with a goal of 10s.	[AD9]
AONGCREQ-012	At least one ROI (windowing/regions of interest) per detector should be supported, as a goal multiple ROIs.	4.3, [AD9]
AONGCREQ-013	It should be possible to update the ROI definition (at a minimum start coordinates) dynamically when a loop of readouts is in operation.	Not possible in the IR.
AONGCREQ-014	Ability to synchronize the start of a readout sequence with an external trigger to within 30 micro seconds.	4.9, [AD9]
AONGCREQ-015	1x1 to 16x16 defined in steps of 1 pixel binning is the same for each window within a given detector.	4.3, [AD9]
AONGCREQ-024	It should be possible to command the NGC to execute a defined read sequence for the defined mosaic	4.7, [AD9]
AONGCREQ-025	N times.	[AD9]
AONGCREQ-026	with a user definable (in microseconds) delay (from 0 to TBD) between executions	[AD9]
AONGCREQ-027	where N is a value between 1 and Inf.	[AD9]
AONGCREQ-028	A stop command for a given mosaic should stop the current loop of readout sequences at the completion of the next cycle	3.5.1, [AD9]
AONGCREQ-028a	It should be possible to instruct the NGC to pass a 15bit data ramp over the real time data link in a standard frame including both start and end of frame words at frame rates up to the maximum supported	TBD - must be supported by hardware
AONGCREQ-029	As a goal it should be possible to load simulated images into the NGC memory and have them ‘played back’ over the RT data link as though coming from a normal readout sequence.	TBD - must be supported by hardware

Item	Requirement	Section
AONGCREQ-030	The replay speed should be a user parameter up to the maximum possible frame rate.	TBD - must be supported by hardware
AONGCREQ-038	Define one (or more) ROIs for a specified detector with start pixel and window dimensions.	4.3, [AD9]
AONGCREQ-039	Define readout mode for detector mosaic.	4.1, 4.7, [AD9]
AONGCREQ-040	Enable/Disable storage of detector frames.	4.2
AONGCREQ-041	Define sub-sampling of readout to be passed to instrument workstation for visualisation or storage.	4.1, [AD9]
AONGCREQ-042	Start readout of mosaic of detectors for N cycles, 1<=N<=Inf	3.5.1, 3.5.2, [AD9]
AONGCREQ-043	with optional synchronization with external synch signal.	[AD9]
AONGCREQ-044	Stop readout of a mosaic of detectors.	3.5.1, [AD9]
AONGCREQ-045	Acquire and store a contiguous set of N frames from a ‘group’ of detectors.	4, [AD9]
AONGCREQ-046	Upload a set of detector frames to be replayed in simulation mode.	TBD - must be supported by hardware
AONGCREQ-047	Replay previously uploaded simulated data frames at defined loop frequency.	TBD - must be supported by hardware

7 APPENDIX

7.1 Database Classes

7.1.1 Sequencer Class

```
CLASS BASE_CLASS ngcdcsSEQ
BEGIN
    // Optional name
    ATTRIBUTE BYTES64 name ““

    // Clock pattern file name
    ATTRIBUTE BYTES256 clkFile ““

    // Program file name
    ATTRIBUTE BYTES256 prgFile ““

    // Status
    ATTRIBUTE BYTES32 statusName “idle”
    ATTRIBUTE INT32 status 0

    // Dwelltime (multiplier and add)
    ATTRIBUTE INT32 timeFactor 1
    ATTRIBUTE INT32 timeAdd 0

    // Sequencer continuous mode flag
    ATTRIBUTE INT32 continuous 0

    // Sequencer trigger mode flag
    ATTRIBUTE INT32 triggerMode 0

    // Sequencer run-control flag
    ATTRIBUTE INT32 runCtrl 1

    // Read-out window
    ATTRIBUTE INT32 startX 1
    ATTRIBUTE INT32 startY 1
    ATTRIBUTE INT32 nx 0
    ATTRIBUTE INT32 ny 0
END
```

7.1.2 CLDC Class

```
CLASS BASE_CLASS ngcdcsCLDC
BEGIN
    // Optional name
    ATTRIBUTE BYTES64 name ““

    // Configuration file name
    ATTRIBUTE BYTES256 cfgFile ““

    // Status
    ATTRIBUTE BYTES32 statusName “disabled”
    ATTRIBUTE INT32 status 0

    // Clock monitor 1
    ATTRIBUTE INT32 clkMon1 1

    // Clock monitor 2
    ATTRIBUTE INT32 clkMon2 1

    // clock-voltage settings of current CLDC-board (Volt)
    ATTRIBUTE Table clk(16,
                        BYTES64 nameLow,
                        BYTES64 nameHigh,
                        DOUBLE voltageLow,
                        DOUBLE voltageHigh,
                        DOUBLE range1Low,
                        DOUBLE range1High,
                        DOUBLE range2Low,
                        DOUBLE range2High,
                        INT32 connected,
                        INT32 reserved)

    // DC-voltage settings of current CLDC-board (Volt)
    ATTRIBUTE Table dc(20,
                       BYTES64 name,
                       DOUBLE voltage,
                       DOUBLE range1,
                       DOUBLE range2,
                       INT32 connected,
                       INT32 reserved)
END
```

7.1.3 ADC Class

```
CLASS BASE_CLASS ngcdcsADC
BEGIN
    // Optional name
    ATTRIBUTE BYTES64 name ““

    // Total number of ADC units on this module
    ATTRIBUTE INT32 num 4

    // Number of bits per pixel
    ATTRIBUTE INT32 bitPix 16

    // Number of enabled ADC units on this module
    ATTRIBUTE INT32 enable 0

    // Conversion strobe delay in ticks
    ATTRIBUTE INT32 delay 0

    // Packet size
    ATTRIBUTE INT32 packetSize 4

    // Packet routing length (number of packets from down-link)
    ATTRIBUTE INT32 packetCnt 0

    // Conversion strobe 1 (enable/disable)
    ATTRIBUTE INT32 convert1

    // Conversion strobe 2 (enable/disable)
    ATTRIBUTE INT32 convert2

    // Operational mode
    ATTRIBUTE INT32 opMode 0

    // Simulation mode
    ATTRIBUTE INT32 simMode 0

    // Monitor channel 1
    ATTRIBUTE INT32 monitor1 1

    // Monitor channel 2
    ATTRIBUTE INT32 monitor2 1

    // Offset per group
    ATTRIBUTE Vector offset (32, DOUBLE)
END
```

7.1.4 Controller Base Class

```
CLASS BASE_CLASS ngcdcsCTRL
BEGIN
    LOOP 4 ATTRIBUTE ngcdcsSEQ seq_#
    LOOP 8 ATTRIBUTE ngcdcsCLDC cldc_#
    LOOP 16 ATTRIBUTE ngcdcsADC adc_#
END
```

7.1.5 Acquisition Module Class

```
CLASS BASE_CLASS ngcdcsACQ
BEGIN
    // Optional name
    ATTRIBUTE BYTES64 name ““

    // Status
    ATTRIBUTE BYTES32 statusName “idle”
    ATTRIBUTE INT32 status 0

    // Process name
    ATTRIBUTE BYTES256 procName ““

    // Number of bursts
    ATTRIBUTE INT32 burst 0

    // Number of frames to skip before burst
    ATTRIBUTE INT32 burstSkip 0

    // Continuous mode flag (no counter reset)
    ATTRIBUTE INT32 continuous 0

    // Transfer flag
    ATTRIBUTE INT32 transfer 0

    // Table containing names and attributes of all available frames
    ATTRIBUTE Table frame(32,
                          BYTES64 name,
                          INT32 gen,
                          INT32 store,
                          INT32 breakCond,
                          INT32 sx,
                          INT32 sy,
                          INT32 nx,
                          INT32 ny)

    // Host name (NGC-LCU)
    ATTRIBUTE BYTES64 host ““

    // Command port
    ATTRIBUTE INT32 cmdPort 0

    // Data port
    ATTRIBUTE INT32 dataPort 0

    // Acquisition window
    ATTRIBUTE INT32 startX 1
    ATTRIBUTE INT32 startY 1
    ATTRIBUTE INT32 nx 0
    ATTRIBUTE INT32 ny 0
END
```

7.1.6 System Status Class

```
CLASS BASE_CLASS ngcdcsSYSTEM
BEGIN
    // Server name
    ATTRIBUTE BYTES64 serverName ""

    // Server process id
    ATTRIBUTE INT32 serverPid 0

    // Version string
    ATTRIBUTE BYTES256 version ""

    // DET category index
    ATTRIBUTE INT32 detIndex 1

    // System configuration file
    ATTRIBUTE BYTES256 sysCfgFile ""

    // Detector configuration file
    ATTRIBUTE BYTES256 detCfgFile ""

    // Current operation mode
    ATTRIBUTE BYTES32 opMode "NORMAL"

    // System status
    ATTRIBUTE BYTES32 stateName "OFF"
    ATTRIBUTE INT32 state 1
    ATTRIBUTE BYTES32 subStateName "idle"
    ATTRIBUTE INT32 subState 1

    // Alarm
    ATTRIBUTE BYTES256 alarm ""

    // Number of sequencer modules in system
    ATTRIBUTE INT32 numCldcMod 0

    // Number of CLDC modules in system
    ATTRIBUTE INT32 numSeqMod 0

    // Number of ADC modules in system
    ATTRIBUTE INT32 numAdcMod 0

    // Number of acquisition modules in system
    ATTRIBUTE INT32 numAcqMod 0

    // Number of FITS header blocks to reserve
    ATTRIBUTE INT32 fitsHdrSize 0

    // Status-polling flag
    ATTRIBUTE INT32 polling 0

    // Current action for action log
    ATTRIBUTE BYTES256 currentAction ""

    // Table containing the dynamic system parameters
    ATTRIBUTE Table param(128,
                          BYTES32 name,
                          BYTES32 value)
END
```

7.1.7 Exposure Class

```
CLASS inscEXPOSURE ngcdcsEXP
BEGIN
    // Exposure status
    ATTRIBUTE BYTES32 expStatusName “inactive”

    // Path to detector data
    ATTRIBUTE BYTES256 dataPath ““

    // New data file
    ATTRIBUTE BYTES256 newDataFileName ““

    // Exposure base name
    ATTRIBUTE BYTES256 baseName ““

    // Naming scheme used (default request-naming)
    ATTRIBUTE INT32 naming 1

    // Multiple files
    ATTRIBUTE INT32 oneFile 1

    // Data file format
    ATTRIBUTE INT32 format 1

    // Exposure time (seconds)
    ATTRIBUTE INT32 time 0

    // Exposure countdown (seconds)
    ATTRIBUTE INT32 countDown 0

    // Generate extended FITS header
    ATTRIBUTE INT32 extFits 0

END
```

7.1.8 Read-Out Mode Definition Class

```
CLASS BASE_CLASS ngcdcsMODE
BEGIN
    // Current read-out mode (name)
    ATTRIBUTE BYTES64 readModeName ““

    // Current read-out mode (id)
    ATTRIBUTE INT32 readModeId 0

    // Table containing name and id and a short description of
    // all defined read-out modes (terminated with empty string
    // and/or negative id)
    ATTRIBUTE Table readModeList(32,
                                 BYTES64 name,
                                 INT32 id,
                                 BYTES256 desc)
END
```

7.1.9 Guiding Class

```
CLASS BASE_CLASS ngcdcs2AG
BEGIN
    // Offset correction vector + quality
    ATTRIBUTE Vector offset (3, FLOAT)
END
```

7.1.10 Chopper Interface Class

```
CLASS BASE_CLASS ngcdcsCHOPPER
BEGIN
    // Chopper status (on or off)
    ATTRIBUTE INT32 status 0

    // Frequency
    ATTRIBUTE DOUBLE freq 0.0

    // Transition time
    ATTRIBUTE DOUBLE transTime 0.0
END
```

7.1.11 Server Class

```
CLASS ngcdcsCTRL ngcdcsSERVER
BEGIN
    LOOP 32 ATTRIBUTE ngcdcsACQ acq_#
    ATTRIBUTE ngcdcsSYSTEM system
    ATTRIBUTE ngcdcsEXP exposure
    ATTRIBUTE ngcdcsMODE mode
    ATTRIBUTE ngcdcs2AG guiding
END

CLASS ngcdcsSERVER ngcirconSERVER
BEGIN
    ATTRIBUTE ngcdcsCHOPPER chopper
END
```

7.2 Reference

7.2.1 ngcircon Server

NAME

```
       ngcircon - NGCIRSW control server
```

SYNOPSIS

```
       ngcircon [options]
```

DESCRIPTION

```
       NGCIRSW control server. The server is based on the CCS event
       toolkit EVH. Commands can be sent to the server via the
       CCS message system.
```

OPTIONS

```
       -db <point>         - database point (without instance)
                             (default: point = <alias>ngcircon)
       -inst <label>       - server instance label
                             (default: label = )
       -cfg <file-name>    - load system configuration file
       -dcf <file-name>    - detector configuration file
       -sim <LCU|HW>       - start in simulation mode
       -online             - go online after start
       -start              - auto-start at online
       -poll               - enable status polling
       -gui [name]         - launch GUI (name is optional)
                             (default: no GUI)
       -ld <dictionary>    - load dictionary (repetitive)
       -det <index>        - detector category index
                             (default: index = 1)
       -xterm              - start processes in x-terminal
       -verbose <level>    - verbose level
                             (default: level = 0)
       -log <level>        - log level
                             (default: level = 0)
       -shell              - launch command shell
       -help or -usage     - show options
```

ENVIRONMENT

```
       The environment variables INS_ROOT and INS_USER are used to build
       the basic search paths ($INS_ROOT/$INS_USER/...) for configuration
       files unless absolute paths are given. If the INS_USER environment
       variable is not set, then the default value SYSTEM is assumed.
```

COMMANDS

```
       The commands are defined in the command definition table of the
       server (ngcircon.cdt).
```

SEE ALSO

```
       ngcdcsEVH(4), ngcdcsCTRL_CLASS(4), evhTASK(4), EVH(5)
```

7.2.2 Command Definition Table

```
PUBLIC_COMMANDS

COMMAND=          ABORT
FORMAT=           A
PARAMETERS=
     PAR_NAME=    expoId
     PAR_TYPE=    INTEGER
     PAR_OPTIONAL=YES
REPLY_FORMAT=     A
REPLY_PARAMETERS=
     PAR_NAME=    done
     PAR_TYPE=    STRING
     PAR_DEF_VAL= “OK”
HELP_TEXT=
Abort exposure.
@

COMMAND=          BREAK
FORMAT=           A
REPLY_FORMAT=     A
HELP_TEXT=
Interrupt server.
@

COMMAND=          CLDC
FORMAT=           A
PARAMETERS=
     PAR_NAME=    module
     PAR_TYPE=    INTEGER
     PAR_OPTIONAL=YES

     PAR_NAME=    default
     PAR_TYPE=    LOGICAL
     PAR_OPTIONAL=YES

     PAR_NAME=    enable
     PAR_TYPE=    LOGICAL
     PAR_OPTIONAL=YES

     PAR_NAME=    disable
     PAR_TYPE=    LOGICAL
     PAR_OPTIONAL=YES

     PAR_NAME=    zero
     PAR_TYPE=    LOGICAL
     PAR_OPTIONAL=YES

     PAR_NAME=    calibrate
     PAR_TYPE=    LOGICAL
     PAR_OPTIONAL=YES

     PAR_NAME=    clear
     PAR_TYPE=    LOGICAL
     PAR_OPTIONAL=YES

     PAR_NAME=    check
     PAR_TYPE=    LOGICAL
     PAR_OPTIONAL=YES

     PAR_NAME=    restore
     PAR_TYPE=    STRING
     PAR_OPTIONAL=YES

     PAR_NAME=    save
     PAR_TYPE=    STRING
     PAR_OPTIONAL=YES
REPLY_FORMAT=     A
REPLY_PARAMETERS=
     PAR_NAME=    done
     PAR_TYPE=    STRING
     PAR_DEF_VAL= “OK”
HELP_TEXT=
CLDC module interaction. The -module option specifies the
CLDC module the command refers to. Module numbers start with 1.
A zero module number refers to all modules. -default will
reset all voltages to their default values as defined in the
voltage configuration file. -enable/-disable will enable/disable
the output of the referenced module. -zero sets all voltages on
the module to zero. -calibrate performs a calibration which can
be cleared with -clear. -check checks all voltages on the module
against the telemetry. -restore restores the setup value of a
voltage keyword to the value as it was given in the configuration
file. -save saves the current voltage configuration to the given
file.
@

COMMAND=          END
FORMAT=           A
PARAMETERS=
     PAR_NAME=    expoId
     PAR_TYPE=    INTEGER
     PAR_OPTIONAL=YES

REPLY_FORMAT=     A
REPLY_PARAMETERS=
     PAR_NAME=    done
     PAR_TYPE=    STRING
     PAR_DEF_VAL= “OK”
HELP_TEXT=
Terminate exposure as quickly as possible with an intermediate result.
@

COMMAND=          EXIT
FORMAT=           A
REPLY_FORMAT=     A
HELP_TEXT=
Make the server exit/terminate.
@

COMMAND=          FRAME
FORMAT=           A
PARAMETERS=
     PAR_NAME=    module
     PAR_TYPE=    INTEGER
     PAR_OPTIONAL=YES

     // frame name
     PAR_NAME=    name
     PAR_TYPE=    STRING
     PAR_OPTIONAL=NO

     // generate
     PAR_NAME=    gen
     PAR_TYPE=    STRING
     PAR_OPTIONAL=YES

     // store
     PAR_NAME=    store
     PAR_TYPE=    STRING
     PAR_OPTIONAL=YES

     // break condition
     PAR_NAME=    break
     PAR_TYPE=    INTEGER
     PAR_OPTIONAL=YES

     // window to be transferred
     PAR_NAME=    win
     PAR_TYPE=    INTEGER
     PAR_OPTIONAL=YES
     PAR_MAX_REPETITION=4

REPLY_FORMAT =    A
REPLY_PARAMETERS=
     PAR_NAME=    done
     PAR_TYPE=    STRING
     PAR_DEF_VAL= “OK”
HELP_TEXT =
Data frame setup. The -module option specifies the acquisition
module the command refers to. Module numbers start with 1. A zero
module number refers to all modules. -name defines the name of
the frame type to which the command refers. -gen/-store define
whether the frame of the given type will be generated and/or
stored to disk. -break defines the number of frames of that type
to be produced until the exposure can terminate (break condition).
-win followed by START-X, START-Y, NX, NY specifies a software
window to be applied for that frame type.
@

COMMAND=          KILL
FORMAT=           A
REPLY_FORMAT=     A
HELP_TEXT=
Send a KILL signal to the server.
@

COMMAND=          MSGDLOG
FORMAT=           A
REPLY_FORMAT=     A
HELP_TEXT=
Disable autologging of messages sent or received by the application.
@

COMMAND=          MSGELOG
FORMAT=           A
REPLY_FORMAT=     A
HELP_TEXT=
Enable autologging of messages sent or received by the application.
@

COMMAND=          NGC
FORMAT=           B
REPLY_FORMAT=     A
HELP_TEXT=
Issue a NGC command. This is a low-level interface to all
NGC hardware functions.
@
```

```
COMMAND=          OFF
FORMAT=           A
REPLY_FORMAT=     A
REPLY_PARAMETERS=
     PAR_NAME=    done
     PAR_TYPE=    STRING
     PAR_DEF_VAL= “OK”
HELP_TEXT=
Close all devices and make all sub-processes terminate.
@

COMMAND=          ONLINE
FORMAT=           A
REPLY_FORMAT=     A
REPLY_PARAMETERS=
     PAR_NAME=    done
     PAR_TYPE=    STRING
     PAR_DEF_VAL= “OK”
HELP_TEXT=
Bring server to on-line state.
@

COMMAND=          PING
FORMAT=           A
REPLY_FORMAT=     A
REPLY_PARAMETERS=
     PAR_NAME=    done
     PAR_TYPE=    STRING
     PAR_DEF_VAL= “OK”
HELP_TEXT=
Make a check of the functioning of the server and send back an
overall status message.
@

COMMAND=          RESET
FORMAT=           A
REPLY_FORMAT=     A
REPLY_PARAMETERS=
     PAR_NAME=    done
     PAR_TYPE=    STRING
     PAR_DEF_VAL= “OK”
HELP_TEXT=
Reset controller front-end.
@

COMMAND=          SELFTST
FORMAT=           A
PARAMETERS=
     PAR_NAME=    function
     PAR_TYPE=    STRING
     PAR_OPTIONAL=YES
     PAR_MAX_REPETITION=999

     PAR_NAME=    repeat
     PAR_TYPE=    INTEGER
     PAR_OPTIONAL=YES
     PAR_DEF_VAL= 1

REPLY_FORMAT =    A
REPLY_PARAMETERS=
     PAR_NAME=    done
     PAR_TYPE=    STRING
     PAR_DEF_VAL= "OK"
HELP_TEXT =
Execute a selftest (hardware and software) of the specified
function(s). -repeat specifies, how often the given test is
repeated in a loop. Valid functions are SEQ, CLDC, ADC and ACQ.
If no function is given, then an overall selftest of all
functions is performed.
@

COMMAND=          SEQ
FORMAT=           A
PARAMETERS=
     PAR_NAME=    module
     PAR_TYPE=    INTEGER
     PAR_OPTIONAL=YES

     PAR_NAME=    save
     PAR_TYPE=    STRING
     PAR_OPTIONAL=YES

     PAR_NAME=    tmo
     PAR_TYPE=    INTEGER
     PAR_OPTIONAL=YES
     PAR_MAX_REPETITION=2

     PAR_NAME=    wait
     PAR_TYPE=    STRING
     PAR_OPTIONAL=YES

     PAR_NAME=    stop
     PAR_TYPE=    LOGICAL
     PAR_OPTIONAL=YES

     PAR_NAME=    start
     PAR_TYPE=    LOGICAL
     PAR_OPTIONAL=YES

     PAR_NAME=    step
     PAR_TYPE=    LOGICAL
     PAR_OPTIONAL=YES

     PAR_NAME=    trigger
     PAR_TYPE=    LOGICAL
     PAR_OPTIONAL=YES

REPLY_FORMAT=     A
REPLY_PARAMETERS=
     PAR_NAME=    done
     PAR_TYPE=    STRING
     PAR_DEF_VAL= "OK"
HELP_TEXT=
Sequencer module interaction. The -module option specifies the
sequencer module the command refers to. A zero module number
refers to all modules. -start starts the sequencer and the
associated acquisition processes. -stop stops the sequencer
and the associated acquisition processes. -step starts sequencer
and acquisition and lets the sequencer run till next breakpoint.
-save saves the current clock pattern configuration to the given
file. -tmo specifies a timeout (in seconds) for the sequencer wait
instruction. The second parameter of this function gives a polling
interval in milliseconds (default is 100). -wait waits for the
specified sequencer program event (trigger|break|end) to occur.
Module numbers start with 1. -trigger issues a software trigger.
```

```
@

COMMAND=          SETUP
FORMAT=           A
PARAMETERS=
     PAR_NAME=    expoId
     PAR_TYPE=    INTEGER
     PAR_OPTIONAL=YES

     PAR_NAME=    file
     PAR_TYPE=    STRING
     PAR_OPTIONAL=YES
     PAR_MAX_REPETITION=999

     PAR_NAME=    function
     PAR_TYPE=    STRING
     PAR_OPTIONAL=YES
     PAR_MAX_REPETITION=999

     PAR_NAME=    default
     PAR_TYPE=    LOGICAL
     PAR_OPTIONAL=YES

REPLY_FORMAT =    A
REPLY_PARAMETERS=
     PAR_NAME=    done
     PAR_TYPE=    STRING
     PAR_DEF_VAL= “OK”
HELP_TEXT =
Setup the functions as listed. The -default flag sets all
parameters to their default values as specified in the
parameter default setup file. -file loads a setup from the
given file.
@

COMMAND=          SIMULAT
SYNONYMS=         SIM
FORMAT=           A
PARAMETERS=
     PAR_NAME=    function
     PAR_TYPE=    STRING
     PAR_OPTIONAL=YES
REPLY_FORMAT=     A
REPLY_PARAMETERS=
     PAR_NAME=    done
     PAR_TYPE=    STRING
     PAR_DEF_VAL= “OK”
HELP_TEXT=
Switch to simulation mode. If function is LCU then all processes
will be launched on the local host and all HW is simulated. If no
function is specified or function is set to HW, then only the
hardware is simulated. Other values for function may be used to
set additional subsystems to simulation mode.
@

COMMAND=          STANDBY
FORMAT=           A
REPLY_FORMAT=     A
REPLY_PARAMETERS=
     PAR_NAME=    done
     PAR_TYPE=    STRING
     PAR_DEF_VAL= “OK”
HELP_TEXT=
Bring server to stand-by state. Sub-processes are running, but
the physical connection to the hardware is closed.
@

COMMAND=          START
FORMAT=           A
PARAMETERS=
     PAR_NAME=    expoId
     PAR_TYPE=    INTEGER
     PAR_OPTIONAL=YES

     PAR_NAME=    at
     PAR_TYPE=    STRING
     PAR_OPTIONAL=YES
     PAR_DEF_VAL= “now”
REPLY_FORMAT=     A
REPLY_PARAMETERS=
     PAR_NAME=    done
     PAR_TYPE=    STRING
     PAR_DEF_VAL= “OK”
HELP_TEXT=
Start new exposure. -at defines a start time (UTC) in the
ISO time format hh:mm:[ss[.uuuu]]. If -at is not present or
contains the value <now>, then the exposure is started
immediately.
@

COMMAND=          STATUS
FORMAT=           A
PARAMETERS=
     PAR_NAME=    expoId
     PAR_TYPE=    INTEGER
     PAR_OPTIONAL=YES

     PAR_NAME=    function
     PAR_TYPE=    STRING
     PAR_OPTIONAL=YES
     PAR_MAX_REPETITION=999

REPLY_FORMAT =    A
HELP_TEXT =
Get status for various functions.
@

COMMAND=          STOPSIM
FORMAT=           A
REPLY_FORMAT=     A
REPLY_PARAMETERS=
     PAR_NAME=    done
     PAR_TYPE=    STRING
     PAR_DEF_VAL= “OK”
HELP_TEXT=
Switch to normal operation mode.
@

COMMAND=          VERBOSE
FORMAT=           A
PARAMETERS=
     PAR_NAME=    on
     PAR_TYPE=    LOGICAL
     PAR_OPTIONAL=YES

     PAR_NAME=    off
     PAR_TYPE=    LOGICAL
     PAR_OPTIONAL=YES
REPLY_FORMAT=     A
HELP_TEXT=
Switch verbose mode on/off.
@

COMMAND=          VERSION
FORMAT=           A
PARAMETERS=
REPLY_FORMAT=     A
HELP_TEXT=
Return the actual server version.
@

COMMAND=          WAIT
FORMAT=           A
PARAMETERS=
     PAR_NAME=    expoId
     PAR_TYPE=    INTEGER
     PAR_OPTIONAL=YES
REPLY_FORMAT=     A
REPLY_PARAMETERS=
     PAR_NAME=    expStatus
     PAR_TYPE=    INTEGER
     PAR_DEF_VAL= 0
HELP_TEXT=
Wait for exposure to finish. The command immediately
returns an intermediate reply indicating the current exposure
status. The last reply is sent, when the exposure has finished.
@

MAINTENANCE_COMMANDS

TEST_COMMANDS

// --- oOo ---
```

7.2.3 Control Server Class

NAME

```
       ngcdcsEVH - NGC engineering server base class
```

SYNOPSIS

```
       #include <ngcdcsEVH.h>

       ngcdcsEVH server();
```

PARENT CLASS

```
       ngcdcsEVH: public evhTASK, public ngcdcsSRV
```

DESCRIPTION

```
       NGC engineering server base class. The server is based on the
       CCS event toolkit EVH. Commands can be sent to the server via
       the CCS message system.
```

PUBLIC METHODS

```
       ngcdcsEVH();
           Constructor method (use default controller).

       ngcdcsEVH(ngcdcsCTRL *controller);
           Constructor method with specific controller.

       virtual ~ngcdcsEVH();
           Destructor method.

       const char *Version();
           Returns the current version string.

       void SysCfgDefault(const char *cfg);
           Specify a default system configuration file to be used if no
           explicit configuration is requested via command line. By default
           no configuration file is applied but a hard-coded configuration
           for one single main-board and one acquisition module is used.

       void DbPointDefault(const char *point);
           Specify a default database point to be used if no explicit point
           is requested via command line. By default the database point is
           set to <alias>ngcdcs. In any case the instance label will be added
           to the database point name. The actual point to be used is always
           stored in the public data member <dbPoint>.

       void PrintUsage();
           Prints out all server command line options.

       ccsCOMPL_STAT ParseArguments(int argc, char **argv);
           Parses the server command line for additional arguments.

       int Xterm();
           Returns 1 in case sub-processes should start in a separate
           window.

       ngcb_vb_t VerboseHandler();
           Returns the actual verbose message handler.

       int VerboseLevel();
           Returns current verbose-level.

       int LogLevel();
           Returns current log-level.

       void Verbose(const char *format, ...);
       void Verbose(int level, const char *format, ...);
           Verbose method. A message is put on the standard output depending
           on the current verbose level. The message is also logged depending
```

```
        on the current log level. If the verbose level is zero, the message
        is not printed out. If log level is zero the message is not logged.
        The second version additionally compares with a given <level> to
        be exceeded.

    void Log(const char *format, ...);
    void Log(int level, const char *format, ...);
        Log method. The message logged depending on the current log
        level. If the log level is zero the message is not logged.
        The second version additionally compares with a given <level> to
        be exceeded.

    void AddVerbose(const char *format, ...);
        Verbose method. The message is always printed out.

    void AddLog(const char *format, ...);
        Log method. The message is always logged.

    virtual void VerboseCB(const char *msg);
        Verbose output call-back.

    virtual void LogCB(const char *msg);
        Log output call-back.

    ccsCOMPL_STAT Initialize();
        Initialize the control server (call-backs, database, ...).

    ccsCOMPL_STAT Abort();
        Abort the control server. This function should be called before
        exiting. It will shutdown all all sub-modules (sequencer, CLDC,
        ADC, acquisition). The sequencer(s) will be stopped, the CLDC
        module(s) will be disabled and the acquisition processes will
        be terminated before. The last step before deleting the HW-modules
        will be a reset. After being aborted the system may be brought up
        again with the Initialize() member function.

    int MainLoop();
        Enter server main loop. The function returns an exit status
        intended to be passed to the exit() function. Exit status zero
        indicates that the MainLoop returned successfully.

    ccsCOMPL_STAT CmdCallback(const char *cmd, evhCB_METHOD2 method);
        Add a command call-back. The given method will be called upon
        reception of the specified command.

    void EnterCB(msgMESSAGE &msg, vltLOGICAL flag=TRUE);
        This function should be called after entering a call-back method.
        If the flag has a TRUE value, this indicates, that the method
        is active (i.e. will change system parameters). If the flag
        has a FALSE value, this indicates, that the function will be
        passive (i.e. the system does not change in case of a successful
        operation).

    ccsCOMPL_STAT ExitCB();
        This function should be called before leaving the call-back method.
        This will occasionally update the database and server internals
        (if something has changed).

    void ExitCB(msgMESSAGE &msg);
        Same as the above function, but this will additionally send back
        the reply message.

    void CleanupCB();
        Cleanup callback internals, which may have been set during
        EnterCB(). When not using the ExitCB(msgMESSAGE &msg) this
        function should be called after the last reply has been sent.

    ccsCOMPL_STAT CheckSysCfg(ngcdcs_syscfg_t cfg);
        Checks the given system configuration structure (range check,
        consistency check, ...). See below for a description of the
        configuration structure.

    ccsCOMPL_STAT CheckDetCfg(ngcdcs_detcfg_t cfg);
        Checks the given detector configuration structure (range check,
        consistency check, ...). See below for a description of the
        configuration structure.

    virtual int LoadSysCfg(ngcdcs_detcfg_t *cfg, const char *name);
        Uploading method for system configuration. The method can be
        overloaded to implement specific file formats. Must return either
        ngcbSUCCESS in case of successful operation or ngcbFAILURE and
        an error message in ErrMsg().

    virtual int LoadDetCfg(ngcdcs_detcfg_t *cfg, const char *name);
        Uploading method for detector configuration. The method can be
        overloaded to implement specific file formats. The configuration
        should be checked with the CheckDetCfg() method before returning
        with success. Must return either ngcbSUCCESS in case of successful
        operation or ngcbFAILURE and an error message in ErrMsg().

    ccsCOMPL_STAT Configure(ngcdcs_syscfg_t cfg);
        Apply a new system configuration. The configuration should have
        been checked with the CheckSysCfg() method before. See below
        for a description of the configuration structure. The Shutdown()
        method is called before the new system configuration is applied.

    void Shutdown()
        Shutdown all devices/modules. This function is called internally
        from within the Configure() method before a new system
        configuration is applied.

    virtual ccsCOMPL_STAT ConfigureCB1(ngcdcs_syscfg_t cfg);
    virtual ccsCOMPL_STAT ConfigureCB2(ngcdcs_syscfg_t cfg);
        A call-back function which is executed from within the Configure()
        method before and after internal configuration is done. By
        overloading this functions application specific system
        configuration may be added. See below for a description of the
        configuration structure.

    virtual void ShutdownCB1();
    virtual void ShutdownCB2();
        A call-back function which is executed from within the Shutdown()
        method before and after shutdown of the internal modules is done.
        The methods should be used to shutdown application specific
        devices/modules. The shutdown must be clean (i.e. does not
        return with an error).

    int FitsBlock();
        Returns the number of FITS-header blocks as proposed by the
        system.

    virtual ccsCOMPL_STAT ExpCB(int errorId,
                                vltUINT32 newState,
                                const char *newFile,
                                const char *errorString);
        Call-back function, which is executed upon an exposure event.
        The new exposure state can be an or’ed combination of the
        following values:

            ngcdcsEXP_NONE (0)
            ngcdcsEXP_INACTIVE (1)
            ngcdcsEXP_PENDING (2)
            ngcdcsEXP_INTEGRATING (4)
            ngcdcsEXP_PAUSED (8)
            ngcdcsEXP_READING_OUT (16)
            ngcdcsEXP_PROCESSING (32)
            ngcdcsEXP_TRANSFERRING (64)
            ngcdcsEXP_COMPL_SUCCESS (128)
            ngcdcsEXP_COMPL_FAILURE (256)
            ngcdcsEXP_COMPL_ABORTED (512)
        If the state has not changed the newState will have the value
        ngcdcsEXP_NONE (0). If <newFile> is not en empty string, then
        a new file with the given name has been created. <newFile> contains
        a full path name. If <errorString> is not an empty string, then
        an error occured. The error logging will already have been done in
        that case, so this has mainly an informative purpose.

    vltUINT32 ExpStatusVal();
        Returns the current exposure state.

    char *ExpStatusString();
        Returns current exposure state as string value.

    vltLOGICAL ExpActive();
        Returns TRUE in case the exposure is active. Otherwise FALSEE is
        returned.

    ngcbPARAM_LIST *ParamList();
        Returns a pointer to the global dynamic parameter list. The
        list is used by both the acquisition modules and the sequencer
        modules. See man-page of ngcdcsACQ(4), ngcdcsSEQ(4) and ngcbPARAM(4)
        for details.

    virtual void ExitSignal(int sig);
        Exit signal handler. The ngcdcsEVH::ExitSignal(sig) instance
        should always be called before adding own stuff to this call-back.

    ccsCOMPL_STAT ReadAddr(ngcbIFC *dev, ngcb_route_t route, int address,
                           int *buffer, int size);
        Reads size words from the given device address into buffer.
        The ngcb_route_t structure contains the following elements:

          int numHdr;             - Number of headers to target including
                                    the terminating <0x2>
          int hdr[ngcbMAX_MOD];   - Array of headers including the
                                    terminating <0x2>

    ccsCOMPL_STAT WriteAddr(ngcbIFC *dev, ngcb_route_t route, int address,
                            int *buffer, int size);
        Writes size words from buffer to the given device address.
        The ngcb_route_t structure contains the following elements:

          int numHdr;             - Number of headers to target including
                                    the terminating <0x2>
          int hdr[ngcbMAX_MOD];   - Array of headers including the
                                    terminating <0x2>

    ccsCOMPL_STAT WriteBuffer(ngcbIFC *dev, int *buffer, int size);
        Writes a formatted buffer to the given interface device.
        The format of the buffer is:
        <hdr1 hdr2 ... hdrN address data1 data2 ...>

    ccsCOMPL_STAT ReadAddr(int idx, ngcb_route_t route, int address,
                           int *buffer, int size);
    ccsCOMPL_STAT WriteAddr(int idx, ngcb_route_t route, int address,
                            int *buffer, int size);
    ccsCOMPL_STAT WriteBuffer(int idx, int *buffer, int size);
        Same as the above functions, but using the device instance
        number instead of the device pointer itself. The device indices
        start from zero.

    ngcdcsCLDC *CldcMod(ngcbIFC *dev, ngcb_route_t route);
        Module creation hook for the CLDC module. An instance of a
        class derived from the ngcdcsCLDC class must be returned.
        The device and the route should be passed to the underlying
        constructor. In case of failure a NULL pointer should be returned
        and an error should be added to the error-stack.

    ngcdcsSEQ *SeqMod(ngcbIFC *dev, ngcb_route_t route, ngcbPARAM_LIST *list);
        Module creation hook for the sequencer module. An instance of a
        class derived from the ngcdcsSEQ class must be returned.
        The device and the route should be passed to the underlying
        constructor together with a pointer to a dynamic parameter list
        (for details see man-page of the ngcbSEQ(4) and ngcbPARAM(4)
        classes). In case of failure a NULL pointer should be returned
        and an error should be added to the error-stack.

    ngcdcsADC *AdcMod(ngcbIFC *dev, ngcb_route_t route, int num);
        Module creation hook for the ADC module. An instance of a
        class derived from the ngcdcsADC class must be returned.
        The device and the route should be passed to the underlying
        constructor together with the number of ADC groups in this
        module. In case of failure a NULL pointer should be returned
        and an error should be added to the error-stack.

    ngcdcsACQ *AcqMod(ngcdcs_acq_cfg_t cfg, ngcbPARAM_LIST *list);
        Module creation hook for the acquisition module. An instance of a
        class derived from the ngcdcsACQ class must be returned.
        A pointer to a dynamic parameter list must be passed to
        the underlying constructor (for details see man-page of the
        ngcdcsACQ(4) and ngcbPARAM(4) classes). In case of failure a NULL
        pointer should be returned and an error should be added to the
        error-stack.

    virtual int PostProcCB(void *buffer, ngcdcs_finfo_t *finfo, char *erms);
        Post processing call-back, which is executed whenever a new frame
        has been received. The ngcdcs_finfo_t structure <finfo> contains
        all information for the <buffer> and has the following members:

          int type;          - Unique frame type
          char name[64];     - Unique frame name
          int fcnt;          - Frame counter
          int scaleFactor;   - Scaling factor to be applied to normalize
          int bitPix;        - Bits per pixel as defined in the FITS-standard
          int sx;            - Lower left corner (x-direction)
          int sy;            - Lower left corner (y-direction)
          int nx;            - Dimension in x-direction
          int ny;            - Dimension in y-direction
          double crpix1;     - Reference pixel in x-direction
          double crpix2;     - Reference pixel in y-direction
          int detIdx;        - Detector index (for mosaics)
          int expCnt;        - Exposure counter for this type
          char utc[64];      - Time when frame was ready in the pre-processor
          ngcdcsCUBE *cube;  - Data cube object to be used for storing to a cube

        The ngcdcsCUBE class contains the following:

          FILE *fd;            - File descriptor
          int naxis1;          - Dimension in x-direction
          int naxis2;          - Dimension in y-driection
          int naxis3;          - Number of images
          int bitPix;          - Bits per pixel as defined in the FITS-standard
          char fileName[256];  - Actual filename (full path)
          char frameName[64];  - Frame type name for all images in the cube
          int Size();          - Return current cube size
          Close();             - Close the cube
          int Open(const char *path, const char *name); - Open the cube

        Type and dimension should be cross checked for consistency with
        the stored values in <cube>, before adding a frame to a cube.
        The post-processing call-back may return one of the following:

          ngcbSUCCESS - Successful operation
          ngcbFAILURE - Failure (an error should be added to the error-stack)
          ngcbSKIP    - Successful operation - but skip all further
                        actions on the frame (no storage to file,...)

    ccsCOMPL_STAT OnlineCB1();
    ccsCOMPL_STAT OnlineCB2();
    ccsCOMPL_STAT StanbbyCB1();
    ccsCOMPL_STAT StandbyCB2();
    ccsCOMPL_STAT OffCB1();
    ccsCOMPL_STAT OffCB2();
        State switching hook before (1) and after (2) internal
        state-switching.

    ccsCOMPL_STAT SetupCB1(char **list, vltINT32 *size);
        Call-back function, which is executed before the internal setup
        is done. The setup list contains pairs of parameter names and
        values. The list has to be examined here. Application specific
        parameters have to be removed from the list as the internal
        setup handler would report an error for those. So the list and
        size may be modified. Parameters to be handled after the internal
        setup has been done, must nevertheless be removed from the list
        and have to be kept in the overloading class in order to be
        processed afterwards in the SetupCB2().

    ccsCOMPL_STAT SetupCB2();
        Call-back function, which is executed after the internal setup
        has been done. The functions to be executed have to be extracted
        in the SetupCB1() method.

    virtual int LookupCB(const char *name, char *value);
        Callback for parameter lookup. The function must return
        non-zero in case the parameter given by its <name> has
        been resolved and the value had been properly set. Otherwise
        zero must be returned. The value should contain the properly
        formatted data without unit and without comment.
```

PUBLIC DATA MEMBERS

```
char dbPoint[256];  - actual database point name

int state;          - Server state. This is one of

                        ngcdcsSTATE_OFF (evhSTATE_OFF)
                        ngcdcsSTATE_LOADED (evhSTATE_LOADED)
                        ngcdcsSTATE_STANDBY (evhSTATE_STANDBY)
                        ngcdcsSTATE_ONLINE (evhSTATE_ONLINE)

int subState;       - Server sub-state. This is one of

                        ngcdcsSUBSTATE_IDLE (1)
                        ngcdcsSUBSTATE_BUSY  (2)
                        ngcdcsSUBSTATE_ACTIVE (6)
                        ngcdcsSUBSTATE_ERROR (8)

ngcdcsACQ *acq[ngcdcsACQ_MAX_PROC]; - acquisition module instances
int numAcqMod;                      - number of module instances

ngcdcsCTRL *ctrl;        - Instance of the controller class. See man page
                           of ngcdcsCTRL(4) for details.

int readModeId;          - Current read-mode id

char readModeName[64];   - Current read-mode name

ngcbDIC dictionary;      - Object holding the parameter specifications
                           from all loaded dictionaries.

int detIndex;            - Detector category index

ngcdcs_syscfg_t sysCfg;  - Current system configuration

ngcdcs_detcfg_t detCfg;  - Current detector configuration

char opMode[32];         - Current operation mode. This a string value
                           containing either "HW-sim" "LCU-sim" or
                           "normal". It corresponds to the value of the
                           sim flag (0,1,2) in the sysCfg structure.
```

```
int polling;             - Enable/Disable polling of sub-system status
                           (default is 0 - disabled).

int pollInterval;        - Polling interval (in ms). Default is 1000 ms.

The system configuration structure ngcdcs_sys_cfg_t contains
the following members:

    Interface device system configuration:
    ngcdcs_dev_cfg_t dev[ngcdcsCTRL_MAX_DEV];

    CLDC system configuration:
    ngcdcs_cldc_cfg_t cldc[ngcdcsCTRL_MAX_CLDC];

    Sequencer system configuration:
    ngcdcs_seq_cfg_t seq[ngcdcsCTRL_MAX_SEQ];

    ADC-module system configuration:
    ngcdcs_adc_cfg_t adc[ngcdcsCTRL_MAX_ADCMOD];

    Acquisition module system configuration:
    ngcdcs_acq_cfg_t acq[ngcdcsACQ_MAX_PROC];

    int numDev;           - Number of interface devices
    int numCldcMod;       - Number of CLDC-modules
    int numSeqMod;        - Number of sequencer-modules
    int numAdcMod;        - Number of ADC-modules
    int numAcqMod;        - Number of acquisition modules*
    int sim;              - Default operation in simulation mode
    int fitsBlock;        - Number of FITS-blocks to reserve for merging
    int extFits;          - Generate extended FITS-header
    int fileFormat;       - Default file format
    int naming;           - Default naming scheme
    int oneFile;          - All in one file (FITS extension format only)
    int autoOnline;       - Go on-line after start-up
    int autoStart;        - Start at on-line
    char detCfgFile[256]; - Default detector configuration file
    char fileName[256];   - Optional filename
    char dsup[256];       - Optional default parameter setup file
    int polling;          - enable/disable polling of sub-system status
    int pollInterval;     - Polling interval in ms

The interface device configuration structure ngcdcs_dev_cfg_t contains
the following members:

    char name[128]; - Device name
    char env[32];   - Environment name
    char host[64];  - Host where physical interface resides
    char srv[64];   - Optional driver interface process name
    char type[32];  - Optional interface type

The CLDC module system configuration structure ngcdcs_cldc_cfg_t
contains the following members:

    int devIdx;          - Device index
    ngcb_route_t route;  - Route to module
    int autoEnable;      - Enable at on-line
    double margin;       - Margin for voltage check (in volts)
    double telemetryGain; - Gain factor for telemetry
    char name[64];       - Optional name

The sequencer module system configuration structure ngcdcs_seq_cfg_t
contains the following members:

    int devIdx;          - Device index
    ngcb_route_t route;  - Route to module
    char name[64];       - Optional name
```

```
The ADC module system configuration structure ngcdcs_adc_cfg_t
contains the following members:

    int devIdx;           - Device index
    ngcb_route_t route;   - Route to module
    int num;              - Number of ADC groups on board
    int bitPix;           - Number of bits per pixel (16, 18, ...)
    char name[64];        - Optional name

The acquisition module system configuration structure ngcdcs_acq_cfg_t
contains the follwoing members:

    int cmdPort;          - Optional command server port (default is zero)
    int dataPort;         - Optional data server port (default is zero)
    int numDataClient;    - Number of data clients
    int transferMode;     - Data transfer mode. Can be one of

                            ngcppTMODE_VIDEO   - video transfer for RTD
                            ngcppTMODE_SCIENCE - transfer to file

    char host[64];        - Acquisition process host
    char dev[128];        - Acquisition  process DMA device name
    int seqIdx;           - Associated sequencer instance

See man page of ngcbIFC(4) for a definition of the ngcb_route_t structure.

The detector configuration structure ngcdcs_detcfg_t contains the
following members:

    char fileName[256];  - Optional filename for this
    char dsup[256];      - Optional default parameter setup file

    Sequencer module detector configuration:
    ngcdcs_seq_dcf_t seq[ngcdcsCTRL_MAX_SEQ];

    CLDC module detector configuration:
    ngcdcs_cldc_dcf_t cldc[ngcdcsCTRL_MAX_CLDC];

    ADC module detector configuration:
    ngcdcs_adc_dcf_t adc[ngcdcsCTRL_MAX_ADCMOD];

    ngcdcs_chip_t chip[ngcdcsMAX_DET];  - Chip information
    int numDet;                         - Number of detectors in mosaic
    char name[32];                      - Detector system name
    char id[32];                        - Detector system id
    int numOutput;                      - Total number of outputs
    int splitX[ngcdcsMAX_ACQPROC];      - Split to FITS extensions
    int splitY[ngcdcsMAX_ACQPROC];      - Split to FITS extensions
    ngcdcs_rm_t rm[ngcdcsMAX_RM];       - Read-out modes
    int numReadMod;                     - Number of read-out modes
    int rmDefault;                      - Default read-out mode

The sequencer module detector configuration structure ngcdcs_seq_dcf_t
contains the following members:

    char clkFile[256];  - Clock pattern file to load
    int timeFactor;     - Default dwell-time factor
    int timeAdd;        - Default dwell-time add
    int continuous;     - Operate in continuous mode
    int runCtrl;        - External run-control

The CLDC module detector configuration structure ngcdcs_cldc_dcf_t
contains the following members:

    char voltageFile[256]; - Voltage file to load

The ADC module detector configuration structure ngcdcs_adc_dcf_t
contains the following members:
    int delay;         - Default conversion strobe delay
    double offset[32]; - Default offset per group
    int enable;        - Number of enabled ADCs on board
    int simMode;       - Simulation mode
    int opMode;        - Operation mode
    int packetSize;    - Default packet size
    int packetCnt;     - Default packet routing length
    int convert1;      - Conversion on strobe 1
    int convert2;      - Conversion on strobe 2
```

The read-out mode definition structure ngcdcs_rm_t contains the following members:

```
    char name[64];                         - Name
    char desc[256];                        - Description
    char seqPrg[ngcdcsCTRL_MAX_SEQ][256];  - Sequencer programs
    char acqProc[ngcdcsACQ_MAX_PROC][64];  - Processes for acquisition
                                             modules
    char dsup[256];                        - Default parameter setup file
    int id;                                - Unique id
    int hwWin;                             - HW-window supported
```

The chip info structure ngcdcs_chip_t contains the following members:

```
    int posX;        - x-position in mosaic
    int posY;        - y-position in mosaic
    int nx;          - x-dimension (in pixels)
    int ny;          - y-dimension (in pixels)
    int adjustMode;  - Window adjustment mode (0=free, 1=center, ...)
    int adjustX;     - Window adjustment step in x-direction
    int adjustY;     - Window adjustment step in y-direction
    int index;       - Unique index
    double pixSpace; - Pixel to pixel space (meter)
    double pszx;     - Size of pixel in x (um)
    double pszy;     - Size of pixel in y (um)
    double xgap;     - Gap between chips along x (um)
    double ygap;     - Gap between chips along y (um)
    double rgap;     - Angle of gap between chips (deg)
    int rotAngle;    - Rotation angle of chip in mosaic (0, 90, 180, 270)
    int live;        - Detector live (=1) or broken (=0)
    int numOutput;   - Number of output channels
    char type[32];   - Detector type
    char name[32];   - Detector name
    char id[32];     - Another detector identification string
    char date[16];   - Installation date [YYYY-MM-DD]
    int acqIdx;      - Acquisition module index
```

ENVIRONMENT

The environment variables INS_ROOT and INS_USER are used to build the basic search paths ($INS_ROOT/$INS_USER/...) for configuration files unless absolute paths are given. If the INS_USER environment variable is not set, then the default value SYSTEM is assumed.

RETURN VALUES

If not specified differently, all member functions return SUCCESS in case of success. Otherwise FAILURE is returned and an error is added to the error-stack.

SEE ALSO

ngcbIFC(4), ngcbIFC_MSG(4), ngcbPARAM(4), ngcbOBJ(4), ngcbMOD(4), ngcbDIC(4), ngcdcsSEQ_CLASS(4), ngcdcsCLDC_CLASS(4), ngcdcsADC_CLASS(4), ngcdcsACQ_CLASS(4), ngcdcsACQ_DATA_CLASS(4), ngcdcsCTRL_CLASS(4), evhTASK(4)