EUROPEAN SOUTHERN OBSERVATORY

Organisation Européenne pour des Recherches Astronomiques dans l'Hémisphère Austral
Europäische Organisation für astronomische Forschung in der südlichen Hemisphäre

VERY LARGE TELESCOPE

New General detector Controller

Infrared Detector Control Software – User Manual

Document Number: **VLT-MAN-ESO-13660-4085**

Document Issue: **<4>**

Date of Issue: **<27/11/2013>**

	Name	Date	Signature
Author :	Jörg Stegmeier	27.11.2013	
Job Manager :	Gert Finger	27.11.2013	
Releaser:	Gert Finger	27.11.2013	

CHANGE RECORD

ISSUE	DATE	SECTIONS AFFECTED	REASON/INITIATION DOCUMENTS/REMARKS
0.1	15-09-2006	All	First draft
0.2	16-02-2007	3.1 3.2.1, 3.2.2 3.3 4 6.6	Updated figure. New command line options. New section for NGCIRSW operational states. Aligned order with actual CDT. Updated.
0.3	26-05-2007	9.10 16, 17 16.4.4	Burst-Mode Added configuration section. Added read-out mode section.
0.4	30-05-2007	23	Server extensions.
0.5	06-06-2007	12 26.4	Error definitions. Post-processing call-back.
1.0	19-07-2007	All	Minor corrections for first issue.
1.1	24-09-2008	2 4 5 6.2, 6.2, 6.2 9.7 9.6 18 18	Added *ngcrtd* and *ngciracq* SW modules. Some commands added for compatibility with NGCOSW. Wrong reference. Updated due to new HW revisions. Updated FITS header contents. Frame-type parameters. I^2C interface added. Inserted section for shutter interface.
2.0	08-10-2008	None	New issue number.
3.0	03-05-2010	All	Completely revised version.
4.0	27-11-2013	4 10.1.3 23 17.4 24 10.5 9.10 2.4.4	EXEC command. Queue mode. AO-head interface. Multiple data links. SELEX-SAPHIRA interface. Data Merger. Exposure Driver. NGC-LLCU Quick Installation.

Table of Contents

List of Figures

List of Tables

1. Introduction

The software described in this manual is intended to be used in the ESO VLT project by ESO and authorized external contractors only.

While every precaution has been taken in the development of the software and in the preparation of this documentation, ESO assumes no responsibility for errors or omissions, or for damage resulting from the use of the software or of the information contained herein.

1.1. Purpose

This document is the user manual of the New General detector Controller (NGC) software for infrared instruments (*NGCIRSW*) and adaptive optics applications (*AONGCSW*).

Its purpose is to provide people, who intend to use the NGC electronics for infrared instruments or for adaptive optics applications, with all the necessary information to install the SW from scratch, interact programmatically with the SW or operate a camera as a simple standalone instrument.

The manual assumes that the reader has some knowledge of C/C++ and Tcl/Tk languages, UNIX Operating System, VLT Software, in particular CCS. It is not intended to be an introduction to infrared detector systems, and therefore it uses common terminology in this field (e.g. detector reset, readout, detector integration time, etc.) without further explanation.

The control software for optical applications (*NGCOSW*) is described in a separate manual [RD3]. Basically the NGC electronics [RD1] is the same for both infrared and optical applications. Nevertheless there are many differences concerning the usage of the controller and the data acquisition and data handling procedures. To cover both applications in an effective way and also to have a certain backwards compatibility with the predecessors FIERA [RD1] and IRACE [RD14], different SW-architectures have been chosen, which are described in detail in the NGC SW design documents [AD6], [RD1] and [AD7]. The following paragraph summarizes the main differences:

- **Detector Read-Out Schemes**
  For infrared applications the clock-pattern generation is running in an infinite loop and the detector is read-out/reset all the times. The optical detector is read-out just once at the end of an exposure.

- **Data Handling**
  The optical application delivers one frame at the end of the exposure and the only processing to be done is pixel sorting and possibly offset correction (if not yet done by the HW). The infrared data require some pre-processing depending on the read-out mode of the detector in use. The read-out modes, the pre-processing algorithms and the setup-parameters for these algorithms are manifold and require a very high degree of flexibility. The pre-processing task produces an arbitrary number of different result frame types, which all have to be transferred and/or displayed on demand. This also has an impact on the RTD-interface (see section 14) because the frames to be displayed are not necessarily the same as the ones to be stored on disk. The latter is always the case for optical exposures.

- **Exposure Loops**
  For infrared applications "*starting an exposure*" basically means starting to transfer the acquired data to a FITS-file (i.e. the server has to attach to and keep step with a running procedure). The end-of-exposure condition is flexible and depends on both the requested frame types and on the number of frames of each type to be produced and stored. The optical exposure always terminates with the saving of the data, which are read at the end of the exposure and follows a much more rigid scheme ("*wiping*" – "*integrating*" – "*reading-out*" – "*transferring*" – "*completed*"). This scheme implies an active intervention of the control server during the exposure like the application of new voltages in each state and the additional shutter-control, whereas the infrared control server mainly reacts passively on incoming data-frames once the exposure is started. So the demands on process concurrency are very different in both cases.

In order to optimize the commonality of the two systems, the *NGCIRSW* control server can be scaled down to control only the NGC without doing any data acquisition or exposure handling (see section 17.3). In this configuration it can simply be operated as a command driven sub-system of *NGCOSW* to access the controller electronics. The NGC general purpose control server (see section 25) can be used instead of the full infrared version, to keep the system more lightweight.

A conscious effort has been made to maintain a certain degree of backwards compatibility of *NGCIRSW* with *IRACE-SW* [RD14]. Nevertheless no responsibility is assumed if this goal is missed in some areas. Where applicable a hint to the major changes with respect to *IRACE-SW* can be found at the end of each section.

1.2. Scope

Scope of this document is the NGC Control Software for infrared instruments (*NGCIRSW*). It also covers all areas needed to implement the *NGCIRSW* as a command-driven sub-system of other applications (e.g. *NGCOSW*).

1.3. Applicable Documents

The following documents form a part of this document to the extent specified herein. In the event of conflict between the documents referenced herein and the contents of this document, the contents of this document shall be considered as a superseding requirement.

AD1	VLT-LIS-ESO-13660-3908	NGC Project Acronyms
AD2	VLT-LIS-ESO-13660-3907	NGC Project Glossary
AD3	VLT-LIS-ESO-13660-3906	NGC Project Documentation
AD4	VLT-SPE-ESO-13660-3207	NGC Requirements Specification
AD5	VLT-SPE-ESO-13660-3670	NGC Software Requirements
AD6	VLT-SPE-ESO-13660-3836	NGC Base Software – Design Description
AD7	VLT-SPE-ESO-13660-3837	NGC – Infrared Detector Control Software – Design Description
AD8	VLT-PRO-ESO-10000-0228	VLT Software Programming Standards
AD9	VLT-MAN-ESO-17210-0667	Guidelines for Development of VLT Application Software
AD10	VLT-SPE-ESO-17212-0001	VLT Instrumentation Software Specification
AD11	VLT-SPE-ESO-17240-0385	INS Common Software Specification
AD12	VLT-MAN-ESO-17240-1913	Installation Tool for VLT SW Packages
AD13	GEN-SPE-ESO-19400-0794	Data Interface Control Document
AD14	GEN-SPE-ESO-00000-0949	VLT Time Reference System Time
AD15	VLT-PLA-ESO-00000-0007	VLT Software Test Plan
AD16	VLT-MAN-ESO-17200-0642	VLT Common Software – Installation Manual
AD17	VLT-MAN-ESO-17240-2325	INS Common Software – Configuration Tool – User Manual

1.4. Reference Documents

The following documents are referenced in this document.

RD1	VLT-MAN-ESO-13660-4510	NGC User Manual
RD2	VLT-SPE-ESO-13660-3835	NGC – Optical Instruments – High-level Software Design Description
RD3	VLT-MAN-ESO-13660-4086	NGC Optical DCS – User Manual
RD4	VLT-SPE-ESO-16100-3729	SPARTA Adaptive Optics – Specifications for NGC
RD5	VLT-MAN-ESO-17210-0619	CCS – User Manual
RD6	VLT-MAN-ESO-17210-0770	Extended CCS – User Manual
RD7	VLT-MAN-ESO-17210-0771	CCS Event Tool Kit – EVH – User Manual
RD8	VLT-MAN-ESO-17200-0908	Tools for Automated Testing – User Manual
RD9	VLT-MAN-ESO-17210-0690	VLT Software – Graphical User Interface – User Manual
RD10	VLT-MAN-ESO-17240-0866	Real Time Display – User Manual
RD11	VLT-MAN-ESO-17240-4566	RTD for CLIP – User Manual
RD12	VLT-MAN-ESO-17240-5546	rtdcore - User Manual
RD13	VLT-MAN-ESO-13640-1388	FIERA CCD Controller Software – User Manual
RD14	VLT-MAN-ESO-14100-1878	IRACE DCS – User Manual
RD15	VMK-8-0-10-1816	USER MANUAL D3882, SELEX Galileo Infrared Ltd

1.5. Abbreviations and Acronyms

DCR	Detector Control Register
MCR	Multiplexer Control Register
SPI	Serial Peripheral Interface
WRR	Window Reset Register
WDR	Window Data Register

Other abbreviations and acronyms used in the NGC project are listed in [AD1].

1.6. Glossary

The relevant concepts used within the NGC project are listed in [AD2].

1.7. Stylistic Conventions

The following styles are used:

bold in the text, for commands, file names, etc. as they must be typed.

Italic in the text, for parts that have to be substituted with the real content before typing.

`Courier` for examples.

<name> in the examples, for parts that have to be substituted with the real content before typing.

The **bold** and *italic* styles are also used to highlight words.

1.8. Naming Conventions

This implementation follows the naming conventions as outlined in [AD10].

1.9. Problem Reporting/Change Request

The form described in [AD16] shall be used. See also section 28 for further hints on troubleshooting.

2. Installation

It is important that the same versions of all NGC software modules are installed on both the instrument workstation and on the NGC-LLCU(s).

2.1. Software Modules

All software modules are under version control in the ESO "*subversion*" repository.

Name	**Description**
dicNGC	Dictionary common to both infrared and optical systems [AD6], [RD1].
ngcdrv	The device driver for the PCI/PCIe back-end card [AD6].
ngcb	The NGC base software module containing the driver interface library 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 simulation mechanism for the NGC controller. See [AD6] for details.
ngcpp	The DMA data-acquisition and pre-/processing module [AD6].
ngcdcs	The NGC detector control software base module implementing the classes for the NGC controller electronics modules (sequencer, CLDC, ADC) and the interfaces to the data acquisition [AD6]. It also contains the general scripts for system startup and shutdown (see section 3).
ngcgui	An engineering GUI used for direct system interaction and data acquisition. A prototype is shown in [AD7].
ngcircon	The NGC system control module for infrared applications [AD7]. This includes the infrared control server instance and all additional infrared specific device classes (chopper, guiding mode, etc.).
ngciracq	The default acquisition processes for NGC infrared detector systems.
ngcrtd	The NGC RTD application.
ngcirins	Installation module for ***pkgin*** (NGC infrared software).
ngcins	Installation module for ***pkgin*** (NGC infrared and optical software).

Table 1 Software Modules

2.2. First Step

When not working in stand-alone mode then in any case it must be possible to ping the NGC-LLCU from the instrument workstation:

```
ping <hostname of NGC-LLCU>
```

and to execute a remote shell on the NGC-LLCU from the instrument workstation:

```
rsh <hostname of NGC-LLCU> ls
```

When the ***rsh*** command fails one has to add the hostname of the instrument workstation (+ *user*) to the ***.rhosts*** file (in the *home* directory of the same *user*) on the NGC-LLCU.

```
wins.hq.eso.org <username>
```

The ***.rhosts*** file must have *r/w*-permission only for the user (otherwise the *rsh* command will fail).

When working in stand-alone mode and in case there is no network (full stand-alone system) one may need to add the value of ***$HOST*** to ***localhost*** in ***/etc/hosts***:

```
127.0.0.1 localhost loopback localhost.localdomain <$HOST> <$HOST>.<domain-name>
```

Example:

```
127.0.0.1 localhost loopback localhost.localdomain wngc wngc.hq.eso.org
```

In stand-alone mode it also must be possible to execute a remote shell on the own host:

```
rsh $HOST ls
```

Also in this case an entry like "***wngc.hq.eso.org <username>***" will be required in the ***.rhosts*** file on the NGC-LLCU.

2.3. Environment Setup

The following environment variables should be set:

Name	Description
$SVNREPO	URL of the ESO "*subversion*" repository. Typically this is set to: *http://svnhq1.hq.eso.org/p1*
$INTROOT	Root of the installation path for binaries, libraries and include files (e.g. /introoot/<user>).
$INS_ROOT	Root of the instrument path for configuration and setup files (e.g. /insroot/<user>). On the NGC-LLCU this is only needed when working in stand-alone mode.
$RTAPENV	Name of the CCS-environment (not needed for stand-alone operation)
$NGC_PORT	Port number for stand-alone operation.
$NGCPP_DATA	Default data port for the acquisition processes. This is also used by the RTD application (see section 14).
$NGCPP_HOST	Default acquisition process host. This is also used by the RTD application (see section 14).
$NGCPP_DPATH	Path where data is stored to local media on the NGC-LLCU (default: *$INS_ROOT/SYSTEM/DETDATA*). If *$INS_ROOT* is not defined or does not indicate a valid directory then "/tmp" is used as default path.

Table 2 Environment Variables

The port numbers may be any unused port (typically *8001*, *8002* and *8003* if not in use for other purposes).

Furthermore all environment variables used in the system configuration files and in the detector configuration files (see section 17) have to be defined.

The directory "*$INS_ROOT/SYSTEM/Dictionary*" should exist before the SW is installed in order to have the dictionary updated.

2.4. Installation

Even though this is not mandatory it is recommended the have the same NGC-SW versions installed on both the instrument workstation the NGC-LLCU.

2.4.1. "subversion" Repository

All NGC software modules are stored in the ESO "*subversion*" repository. The repository is located at the URL:

"http://svnhq1.hq.eso.org/p1"

The URL is typically stored in the environment variable ***$SVNREPO***:

"export SVNREPO=http://svnhq1.hq.eso.org/p1"

The procedure to install the software consists of the following two steps:

1. Retrieve the NGC software module from the SVN repository:

   The latest revision:

   ```
   svn co $SVNREPO/trunk/Detectors/NGC/PKG/NGC
   ```

   Or a given revision:

   ```
   svn co -r <revision> $SVNREPO/trunk/Detectors/NGC/PKG/NGC
   ```

   Or a tagged release:

   ```
   svn co $SVNREPO/tags/RC/NGC/<tag>/NGC
   ```

2. Install all modules:

   ```
   cd NGC/
   make clean man all install
   ```

Caution:
This installation procedure does not include the device driver installation on the NGC-LLCU as this step needs to be done manually with root privileges (system administrator). See section 2.4.3 for the device driver installation instructions.

2.4.2. Installation with pkgin

The ***ngcirins*** software module contains a *pkgin* installation-configuration for the NGC infrared:

```
Retrieve latest or fixed revision:

svn co [-r <revision>] $SVNREPO/trunk/Detectors/NGC/PKG/NGC
svn co [-r <revision>]
$SVNREPO/trunk/Detectors/NGC/PKG/ngcirins

Retrieve tagged release:

svn co $SVNREPO/tags/RC/NGC/<tag>/NGC
svn co $SVNREPO/tags/RC/NGC/<tag>/ngcirins

Build the software:

pkginBuild ngcirins
```

The ***ngcins*** software module contains a *pkgin* installation-configuration for both NGC infrared and optical software:

```
Retrieve latest or fixed revision:

svn co [-r <revision>] $SVNREPO/trunk/Detectors/NGC/PKG
cd PKG/

Retrieve tagged release:

svn co $SVNREPO/tags/RC/NGC/<tag>

Build the software:

cd <tag>/
pkginBuild ngcins
```

See [AD12] for details concerning VLTSW package installation.

2.4.3. Device Driver Installation

The NGC device driver module ***ngcdrv*** is retrieved from the ESO "*subversion*" repository with one of the retrieval steps described in section 2.4.1. Then the device driver has to be installed manually on the NGC-LLCU (the installation is not required on the IWS). The later revisions of the ***ngcdrv*** driver module contain a *Makefile* directive "*make driver*" to install the device driver. This will prompt the user for the root password.

- *cd NGC/ngcdrv/src*
- *make driver*
- *[enter root password when being asked]*

There is also an installation script which is internally called with its proper arguments by the "*make driver*" directive:

```
ngcdrvInstall [root directory]
```

When the root directory of the ***ngcdrv*** module is not explicitly given as first command line option of this script then the script has to be executed in the parent directory of the ***ngcdrv*** module. When the script is not executed as *root* then it will also prompt the user to enter the root password in order to complete the device driver installation.

The manual device driver installation procedure is as follows:

- *cd NGC/*
- *cp –r ngcdrv /tmp*
- *[login as root ("su –")]*
- *cd /tmp/ngcdrv/src*
- *make all install*

Now the driver module can be loaded and unloaded using the scripts:

```
/usr/local/bin/ngcdrv_load
/usr/local/bin/ngcdrv_unload
```

To have the driver module installed at boot-time the instruction line

```
/usr/local/bin/ngcdrv_load
```

has to be added to the system file "*/etc/rc.local*".

Caution: Use "*su –*" , "*telnet*" or "*rlogin*" to login as root. This ensures that path and environment variables are properly (re-)set. The frequently used "*su*" (without "-") does not setup a proper root session and the loading of the module (***ngcdrv_load***) may fail with an error message indicating an "*invalid module format*".

2.4.4. NGC-LLCU Quick Installation

The following steps summarize the procedures to quickly install the NGC-SW on the NGC-LLCUs:

- *login to the NGC-LLCU*
- *make sure that no NGC-processes are running anymore*
- *if not yet done create an INTROOT:*
  - *getTemplateForDirectory INTROOT $INTROOT*
- *cd <your sources-directory or simply "/tmp">*
- *svn co <URL of trunk, branch or tag to be installed>*
- *cd NGC*
- *make clean man all install*
- *cd ngcdrv/src*
- *make driver*
- *[enter root password when being asked]*

3. Startup/Shutdown Procedure

3.1. System Architecture

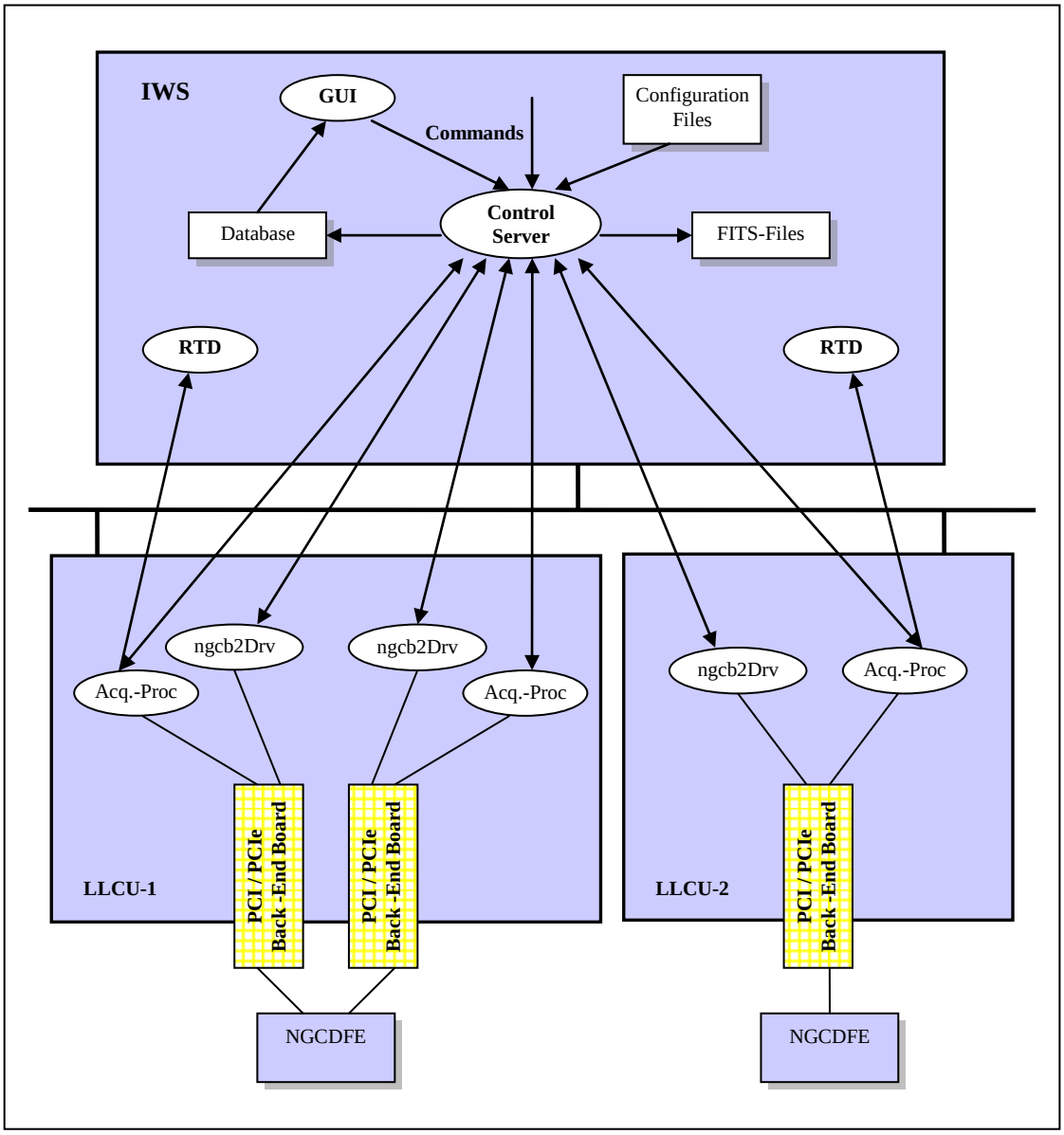

Figure 1 System Architecture

The NGC infrared detector control software (*NGCIRSW*) is running partly on the instrument workstation (IWS) and partly on the NGC-LLCU, where the physical interface(s) to the NGC detector front-end reside (see Figure 1). The NGC-LLCU is a PC running a Linux operating system (kernel 2.4 or higher). In case the maximum nesting depth inside the NGC detector front-end module network has been reached, the system needs to be accessed through additional PCI/PCIe back-end cards. This

may imply the usage of more than one NGC-LLCU, also in case the computing or peripheral bus bandwidth of the NGC-LLCU is not sufficient. So the configuration range covers a simple system controlling one detector via one PCI/PCIe interface card as well as large detector mosaics distributing their data via a huge number of channels among several computers.

The control server communicates with the NGC electronics through driver interface processes (***ngcb2Drv***) running locally on the NGC-LLCUs. The exact mechanism is described in [AD6]. One driver interface process is launched by the control server per physical interface device. The acquisition processes (used for data acquisition and pre-processing) are also launched and controlled by the control server. For maintenance and development operations all processes shown on the IWS side may also run locally on one of the NGC-LLCUs. For software testing and software development all processes may run in simulation mode on the IWS (see section 3.2.3).

The overall controller configuration is done through configuration files in short FITS format as described in more detail in the section 17. The configuration of the controller electronics is divided into system configuration (section 17.1) and detector configuration (section 17.2). The system configuration describes the controller electronics system used (e.g. number and addresses of boards in the system) and also some general system behavior (default file format and naming scheme, etc.). The detector configuration describes the usage of the system with respect to the connected detector(s) (i.e. which clock-patterns and which sequencer programs to load, which voltage setup to apply, etc.). There are cases, where more than one detector is driven by the same controller electronics 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 are kept separate in order to avoid to unnecessarily duplicate the information.

The configuration files are kept in separate instrument specific configuration modules ("***xxdcfg***"), which are version control in the ESO "*subversion*" repository. The configuration modules will take care of installing all files at the proper location (i.e. ***`$INS_ROOT/$INS_USER/COMMON/CONFIGFILES`***). In addition to the system and detector configuration file(s) there are still various other files to be maintained in such a module (e.g. voltage tables, clock pattern definitions, sequencer programs as described in section 16 and also the startup configuration as described in section 3.2.2).

3.2. System Startup

The entry point for the *NGCIRSW* is the control- and data acquisition server (***ngcircon***), through which all system communication is done. The server takes care of starting and shutting down all required sub-processes (except the RTD) as shown in Figure 1.

A general purpose control server (***ngcdcsEvh***, see section 25) is provided in the ***ngcdcs*** SW module. The following sections are also applicable to this basic server by replacing the process name "***ngcircon***" with "***ngcdcsEvh***".

3.2.1. Control Server

The *NGCIRSW* control server is launched with the following command line:

```
ngcircon [-gui [name]] […further options…]
```

This starts the server, and optionally also the graphical user interface (GUI). The following options can be passed to the control server to tune its specific behavior:

-db <point>	Database point to be used (without the instance label and without the <alias>). If this option is not specified, the default database point '<***alias***>***ngcircon***' (or '<***alias***>***ngcdcs***' for ***ngcdcsEvh***) will be used. The instance label (if given) will always be appended to the point-name.
-inst <label>	Server instance label. Used as appendix "**_<label>**" for both the database branch and for the server process name registered with the CCS environment (see section 5). Default is "*no label*" (only one instance running).
-cfg <file-name>	Controller electronics system configuration-file to be loaded by default into this control server instance. Unless an absolute path is given, the file is searched in **$INS_ROOT/$INS_USER/COMMON/CONFIGFILES**. When the file-name is "default", a built-in default configuration is applied (this is only valid for test-purposes). When the option is not present, then the file-name "*NGCIRSW/ngc.cfg*" is used (this only applies to the ***ngciron*** server but not to the general purpose control server which uses the built-in default instead). When the file-name is "none" then the server configures one interface device for system access but no further front-end module (this mode us reserved for board-development only).
-mode <mode>	Operational mode of the detector front-end system after starting up. Valid values are "NORMAL", "LCU-SIM" or "HW-SIM". In HW-simulation mode only the controller electronics functionality is simulated. In LCU-simulation mode additionally all (sub-) processes are started on the local host (i.e. the host where the control server is running) and the acquisition processes start in non real-time mode (no buffer overrun check).
-online	Go automatically to ***ONLINE*** state after startup.
-nocheck	Skip integrity check.
-force	Enforce system startup even when instance check fails.
-start	Automatically start sequencer(s) when going to ***ONLINE*** state.

-poll	Enable sub-system status polling. This will periodically read the controller electronics status.
-pi	Pooling interval in milliseconds. The default value is 1000.
-maxmod <n>	Maximum number of DFE-modules.
-gui [name]	Launch graphical user interface with the specified process name. If no process name is given, but the *–gui* option is set, then the default program ***ngcgui*** will be used.
-ld <dictionary>	Load the given dictionary. The option may be repeated to load more than one additional dictionary. The common dictionary "*ESO-VLT-DIC.NGCDCS*" is always loaded into the system and needs not to be specified using this command line option.
-det <index>	Detector category index (DETi) to be used by higher level SW to forward ***SETUP*** commands to this instance. Default value is 1. The value is just written to the database and has no further operational effect on *NGCIRSW*.
-xterm	Start all (sub-) processes in a separate terminal. This only takes effect, when the verbose level is greater than zero.
–verbose <level>	Verbose level. Prints out system messages to the standard output stream of each (sub-) process. To make them visible for the sub-processes, it is required to start those processes in a separate terminal (*-xterm* option). The level gives the detail of the messages. Default value is 0 (= no verbose output).
-log <level>	Logging level. Logs system messages in the standard log-file, so that they can be seen in the CCS *logMonitor*. The level gives the detail of the messages. Default value is 0 (= no logging).
-pm <0|1>	Enable/disable process monitoring.
-help or –usage	Show available command line options.

Table 3 Control Server Command-Line Options

One server process is launched per instance of *NGCIRSW* (see section 5 for multiple instances). Once the control server is started, the controller electronics system configuration file and the default detector configuration file specified therein are loaded into the server and the state of *NGCIRSW* switches to ***LOADED***. In case the *–online* option is given, the system will automatically go to ***ONLINE*** state (i.e. all sub-processes are started and the overall setup is uploaded to the controller electronics). The operational states of the *NGCIRSW* are described in more detail in section 3.3.

The following command line is used to launch the engineering GUI only (see also section 15):

```
ngcgui [-db <point>][-inst <label>][-cfg <file-name>]
```

The command line options have the same meaning as the respective ones for the control server (see Table 3).

3.2.2. Startup Procedure

The usage of the above command line options is mainly intended for SW-/HW-development and all other cases where a rapid switching between temporary system configurations and operational modes is needed. For a stable and limited set of configurations a more convenient startup procedure is provided, which is based on the common VLTSW configuration tool ("***ctoo***", [AD17]). Using this tool, the startup options for each instance of *NGCIRSW* can be described as a configuration set in short FITS format (e.g. "*xxdcfg<NAME>.cfg*", see example in section 3.2.3):

Keyword	**Type**	**Description**
DET.CON.SERVER	String	Defines the name of the control server to be used.
DET.CON.DATABASE	String	Database point to be used (without the instance label and without the <alias>). If this keyword is not present, the default database point '***<alias>ngcircon***' (or '***<alias>ngcdcs***' for ***ngcdcsEvh***) will be used. The instance label (if given) will always be appended to the point-name.
DET.CON.INSTANCE	String	Defines the instance label for the control server and the database (see section 5). Used as appendix "***_<label>***" for both the database branch and for the server process name registered with the CCS environment. Default is "*no label*" (only one instance running) in case the keyword is not present.
DET.CON.SYSCFG	String	Controller electronics system configuration-file to be loaded by default into this control server instance. Unless an absolute path is given, the file is searched in **$INS_ROOT/$INS_USER/COMMON/CONFIGFILES**. When the file-name is "default", a built-in default configuration is applied (this is only valid for test-purposes). When the option is not present, then the file-name "*NGCIRSW/ngc.cfg*" is used (this only applies to the ***ngciron*** server but not to the general purpose control server which uses the built-in default instead). When the file-name is "none" then the server configures one interface device for system access but no further front-end module (this mode us reserved for board-development only).
DET.CON.OPMODE	String	Defines the operational mode of the detector front-end system after starting up. Valid values are "NORMAL", "LCU-SIM" or "HW-SIM". In HW-simulation mode only the controller electronics functionality is simulated. In LCU-simulation mode additionally all (sub-) processes are started on the local host (i.e. the host where the control server is running) and the acquisition processes start in non real-time mode (no buffer overrun check).
DET.CON.AUTONLIN	Logical	When set to *T*, the detector system automatically goes to ***ONLINE*** state at startup.
DET.CON.AUTOSTRT	Logical	Automatically start sequencer(s) when going to ***ONLINE*** state.
DET.CON.POLL	Logical	Enable sub-system status polling.

Keyword	Type	Description
DET.CON.POLLINT	Integer	Polling interval in milliseconds. The default value is 1000.
DET.CON.GUI	String	Defines the name of the control panel (GUI) to be used.
DET.CON.DICT	String	Defines a list of dictionaries to be loaded. The common "*ESO-VLT-DIC.NGCDCS*" is always loaded into the system and needs not to be specified. The entries are separated by white-space. Only the last descriptor of the full dictionary name is needed here (e.g. "NGCDCS STOO_CFG …").
DET.CON.DETIDX	Integer	Detector category index (DETi) to be used by higher level SW to forward ***SETUP*** commands to this instance. Default value is 1 in case the keyword is not present. The value is just written to the database and has no further operational effect on *NGCIRSW*.
DET.CON.XTERM	Logical	Start all (sub-) processes in new terminal. This will only take effect when verbose mode is switched on (i.e. DET.CON.VERBOSE > 0).
DET.CON.VERBOSE	Integer	Verbose level. System messages are printed to the standard output stream of each (sub-) process. To make them visible for the sub-processes, it is required to start those processes in a separate terminal (DET.CON.XTERM = T). The level gives the detail of the messages. Default value is 0 (= no verbose output) in case the keyword is not present.
DET.CON.LOG	Integer	Logging level. System messages are logged in the standard log-file, so that they can be seen in the CCS *logMonitor*. The level gives the detail of the messages. Default value is 0 (= no logging) in case the keyword is not present.
DET.CON.PROCMON	Integer	Process monitoring level. A zero value disables the process monitoring via watchdogs.
DET.CON.MAXMOD	Integer	Maximum number of DFE modules. The keyword can used to limit the allocated database resources. Default (and maximum) value is 24 (=4 fully equipped racks).
DET.CON.SRVPORT	Integer	Server port number for NOCCS stand alone operation

Table 4 Startup Configuration Keywords

Each configuration module ("***xxdcfg***") contains an "*xxdcfgCONFIG.cfg*" file, which assigns a name and some access-right attributes to startup configuration sets (see example in section 3.2.3). The system startup can now simply be done through a startup script by just passing the name of the startup configuration set:

```
ngcdcsStartServer [configuration set name] [options]
```

This will:

- Search the directory *$INS_ROOT/SYSTEM/COMMON/CONFIGFILES* for a file *(*CONFIG.cfg*) containing the given configuration set name (***DET.SETi.NAME*** = "*configuration set name*").
- Load the startup configuration file (***DET.SETi.FILE1***) assigned to that name. The file name is relative to *$INS_ROOT/SYSTEM/COMMON/CONFIGFILES.*
- Convert the ***DET.CON.XXX*** keywords specified in ***DET.SETi.FILE1*** file into command line options.
- Start the server process with these command line options.
- Wait with a default timeout until the server is actually running (i.e. until the server is responding to ***PING*** commands).

The configuration-set names are typically "XXDCS", where "XX" corresponds to the two-letter instrument code.

The command line options as given in Table 3 can still be passed through to the server in order to override the corresponding values in the configuration set keywords (Table 4). If no configuration set is given only the command line options are used.

There are some special options to tune the behavior of the startup tool which are not passed through to the server application:

```
-restart       - restart server/gui if still running
-nowait        - do not wait until server/gui is up
-timeout <ms>  - timeout for ping in milliseconds (default is 3000)
-print         - just print command line to stdout
```

The "*-print*" option can be used to just print out the command line to *stdout* in order to let another script do the actual server launch.

In case the "*-nowait*" option was used for the startup script a further script can be called to wait until the server is running:

```
ngcdcsWaitServer <config.-set name> [-timeout <ms>]
or
ngcdcsWaitServer –server <name> [-inst <label>] [-timeout <ms>]
```

The timeout is given in milliseconds. If no timeout is specified or the timeout is set to zero, then the built-in default timeout values for *exit*/*ping* are used.

If not done through the server command line itself ("*-gui*" option), the control panel can be started in a similar way:

```
ngcdcsStartGui [config.-set name] [options]
```

This will ensure that the GUI is configured properly to match the associated control server instance. Again the "*-print*" option can be used to just print out the command line to *stdout* in order to let another script do the actual launch.

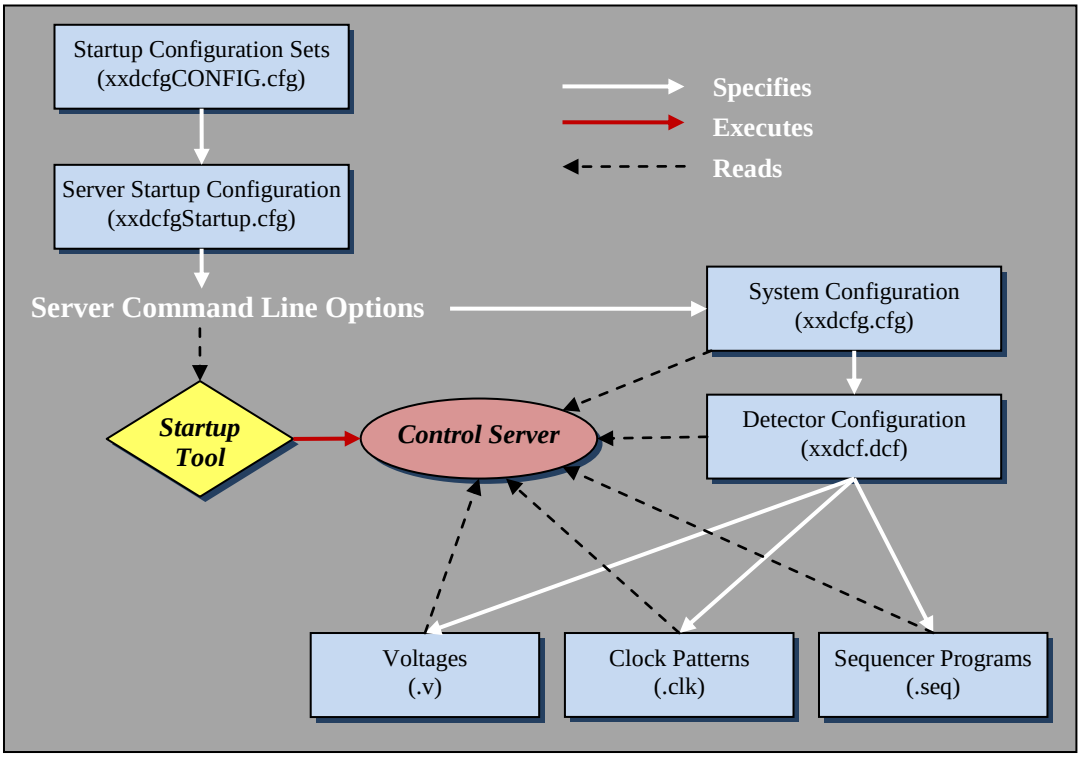

Figure 2 System Startup & Configuration

3.2.3. Simulation Mode

The NGC software can be put into simulation mode with the command:

```
SIMULAT -function HW
```

Where no NGC-LLCU is available all processes can be launched in simulation mode on the instrument workstation:

```
SIMULAT -function LCU
```

The software will return to normal mode with the command:

```
STOPSIM
```

To start the server already in simulation mode the ***DET.CON.OPMODE*** keyword in the startup configuration file can be set accordingly:

```
DET.CON.OPMODE  "HW-SIM"   -HW-Simulation with NGC-LLCU
DET.CON.OPMODE  "LCU-SIM"  -Simulation without NGC-LLCU
DET.CON.OPMOPDE "NORMAL"   -Normal operation
```

Another possibility to "emulate" process simulation ("*LCU-SIM*") on the instrument workstation is to set the ***DET.DEVi.HOST*** and ***DET.ACQi.HOST*** keywords in the system configuration file to "*$HOST*" or to the explicit hostname of the instrument workstation. The ***DET.DEVi.ENV*** keyword should then be set to the IWS-environment (or just "*$RTAPENV*"). The ***DET.DEVi.ENV*** keyword is not needed when the corresponding ***DET.DEVi.TYPE*** keyword is "*socket*" or "*ngc*".

Caution:

*When several NGC-LLCUs are in use then there might arise a port-number conflict when the **DET.ACQi.DATAPORT** keyword is the same for all acquisition module instances (this is legal when all instances run on individual computers). To overcome this problem a special system configuration file for simulation mode should be created with the **DET.ACQi.DATAPORT** keywords being set to different values.*

3.2.4. Examples

Startup Configuration (*xxdcfgCONFIG.cfg*):

```
PAF.HDR.START;
PAF.TYPE          "Configuration";    # Type of PAF
PAF.ID            "@(#) $Id: xxdcfgCONFIG.cfg,v 0.2+ 2006/08/18 13:42:59
vltsccm Exp $";
PAF.NAME          "NGCIRSW";          # Name of PAF
PAF.DESC          "NGCIRSW Startup Configuration";
PAF.CRTE.NAME     "jstegmei";         # Name of creator
PAF.CRTE.DAYTIM   "2006-08-21";       # Civil Time for creation
PAF.LCHG.NAME     "         ";        # Name of person/appl. Changing
PAF.LCHG.DAYTIM   "         ";        # Timestamp of last change
PAF.CHCK.NAME     "         ";        # Name of appl. Checking
PAF.HDR.END;

#
# START CONFIG SET FOR CAMERA 1
# -----------------------------
CONFIG.SET1.NAME     "CAM1";
CONFIG.SET1.DICT     "NGCCON";
CONFIG.SET1.FILE1    "xxdcfgCAM1.cfg";
CONFIG.SET1.PERM1    664; # all
#CONFIG.SET1.PERM1    644; # manager account only
CONFIG.SET1.BACKUP   T;
CONFIG.SET1.LOG      T;

#
# START CONFIG SET FOR CAMERA 2
# -----------------------------
CONFIG.SET2.NAME     "CAM2";
CONFIG.SET2.DICT     "NGCCON";
CONFIG.SET2.FILE1    "xxdcfgCAM2.cfg";
CONFIG.SET2.PERM1    664; # all
#CONFIG.SET2.PERM1    644; # manager account only
CONFIG.SET2.BACKUP   T;
CONFIG.SET2.LOG      T;

#
# ctooConfigArchive CONFIG
# ------------------------
CONFIG.ARCHIVE.NAME      "NGCIRSW";
CONFIG.ARCHIVE.USER      "";
CONFIG.ARCHIVE.MODULE    "xxdcfg";
CONFIG.ARCHIVE.FILE1     "xxdcfg*.cfg";
CONFIG.ARCHIVE.FILE2     "NGCIRSW/*";
CONFIG.ARCHIVE.FILE3     "ngcrtd*.cfg";

# ___oOo___
```

Startup Configuration Set (*xxdcfg<NAME>.cfg*):

```
PAF.HDR.START;
PAF.TYPE          "Configuration";    # Type of PAF
PAF.ID            "@(#) $Id: xxdcfgCAM1.cfg,v 0.2+ 2006/08/18 13:42:59
vltsccm Exp $";
PAF.NAME          "NGCIRSW";          # Name of PAF
PAF.DESC          "NGCIRSW Startup Configuration";
PAF.CRTE.NAME     "jstegmei";         # Name of creator
PAF.CRTE.DAYTIM   "2006-08-21";       # Civil Time for creation
PAF.LCHG.NAME     "         ";        # Name of person/appl. Changing
PAF.LCHG.DAYTIM   "         ";        # Timestamp of last change
PAF.CHCK.NAME     "         ";        # Name of appl. Checking
PAF.HDR.END;

# Control server name
DET.CON.SERVER    "ngcircon"; # "ngcdcsEvh" for general purpose control
server

# Database point
DET.CON.DATABASE "ngcircon"; # "ngcdcs" for general purpose control server

# Instance label for server and OLDB
DET.CON.INSTANCE "";

# HW system configuration file
DET.CON.SYSCFG    "NGCIRSW/ngc.cfg";

# Startup mode (NORMAL, HW-SIM, LCU-SIM)
DET.CON.OPMODE    "HW-SIM";

# Go online after start
DET.CON.AUTONLIN F;

# Auto-start at online
DET.CON.AUTOSTRT F;

# Enable sub-system status polling
DET.CON.POLL      T;

# Detector system index (DETi.XXX)
DET.CON.DETIDX    1;

# Dictionaries to load for this detector system
DET.CON.DICT      "NGCDCS MYDICT …";

# GUI name
DET.CON.GUI       "ngcgui";
```

___oOo___

3.3. NGCIRSW Operational States

The *NGCIRSW* can be in the following operational states (see [AD11]):

State	Description
OFF	The *NGCIRSW* is ***OFF*** when it is not running. Consequently, the *NGCIRSW* can never reply when it is on the ***OFF*** state.
LOADED	The control server is running and is able to respond to commands. The sub-processes are not yet launched and no connection to the controller hardware is established. The database, the controller electronics system configuration file and (if defined therein) the detector configuration file are loaded into the server but are not yet applied to the hardware.
STANDBY	If the control server is not running locally on the NGC-LLCU, then the driver interface process(es) are started on the NGC-LLCU(s). The control server is now ready for use, but the hardware is not yet connected.
ONLINE	The connection to the physical controller device is established, the controller electronics is reset and all modules are configured according to the loaded configuration files. The server reads back all relevant product information (e.g. serial numbers) and checks the consistency with the configuration files. The voltage telemetry of all configured CLDC instances is checked before going to ***ONLINE*** state. The voltage output of all CLDC instances having the ***DET.CLDCi.AUTOENA*** keyword set to '***T***' will be enabled. The acquisition process(es) specified for the default read-out mode are launched on the NGC-LLCU(s). If the ***DET.AUTOSTRT*** keyword has been set to '***T***'', the sequencer(s) and the acquisition process(es) will automatically be started when going to ***ONLINE*** state.

Table 5 Operational States

Figure 3 illustrates the *NGCIRSW* operational states and the commands to switch between them.

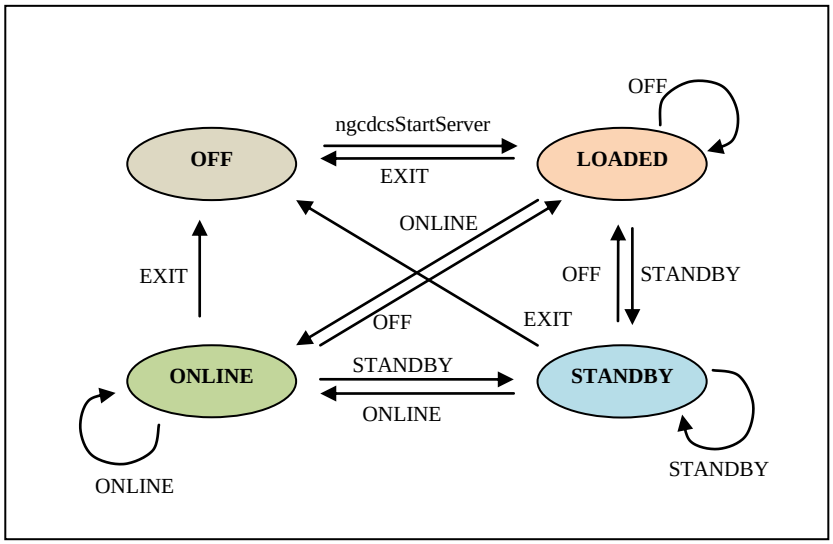

Figure 3 Operational States and State Transitions

Changes with respect to IRACE:

NGCIRSW implements the same operational states as IRACE.

3.4. System Shutdown

The system is shutdown by sending an "***EXIT***" command (see section 4) to the control server (***ngcircon[_<instance-label>]***). The control server will then in turn shutdown all sub-processes. The proper shutdown is also done upon reception of the signals *SIGUSR2* ("***KILL***" command via CCS message system), *SIGINT*, *SIGTERM* (shell command "*kill*") and *SIGHUP*. A shutdown script is available, which waits until the server is actually down (i.e. no more responds to "***PING***" commands):

```
ngcdcsStopServer <config.-set name> [-timeout <ms>]

or

ngcdcsStopServer –server <name> [-inst <label>] [-timeout <ms>]
```

The timeout is given in milliseconds. If no timeout is specified or the timeout is set to zero, then the built-in default timeout values for exit/ping are used.

Changes with respect to IRACE:

The –inst and –cfg command line options are still supported. The location of the configuration files has changed but is transparent for the calling application. Startup and shutdown are backwards compatible when restricting to decimal numbers for the instance label. Environment variables (e.g. host name of NGC-LLCU) can still be used for all entries in the system- and detector configuration file. This still allows the same configuration scheme for server, GUI and RTD as with IRACE.

4. Command Interface

ABORT	Abort exposure immediately. The *–expoId* parameter (integer) may be given, but is not used within this context. The command returns immediately. The "*WAIT*" command must be used to ensure that the exposure is actually terminated.		
AORESET (see also section 23.9)	**-module**	Integer	Specify module index (starts with 1.
	-all	Logical	Reset all sensors.
	-rhum	Logical	Reset the relative humidity sensor.
	-temp	Logical	Reset temperature sensor.
	-pressure	Logical	Reset pressure sensor (dummy function).
	-ovrillum	Logical	Reset over-illumination sensor (dummy function).
	-afe	Logical	Reset analog front-end electronics (AFE).
BREAK	Interrupt server (SIGUSR1).		
CLDC	**-module**	Integer	Module Id (starts with 1). A zero value refers to all modules.
	-default	Logical	Reset all voltages to their default values as defined in the actually loaded voltage configuration file.
	-enable	Logical	Enable CLDC output.
	-disable	Logical	Disable CLDC output.
	-calibrate	Logical	Calibrate voltage channel offsets.
	-clear	Logical	Clear calibration.
	-check	Logical	Check telemetry against setup.
	-restore	String	Restore the setup value of the voltage keyword given in this parameter to the value as it was given in the voltage configuration file.
	-save	String	Save current setup to the file given in this parameter. Unless a full path name is given, the file is stored at the default directory in $INS_ROOT.
	-sverr	String	Saves the current error calibration voltages to the given file.
CONT	**-expoId**	Integer	Ignored.
	-at	String	Defines a continue time (UTC) in the ISO time format hh:mm:[ss[.uuuu]] for a paused exposure. If –at is not present or contains the value <now>, then the exposure is continued immediately.
END	End exposure as quickly as possible with an intermediate result. The *–expoId* parameter (integer) may be given, but is not used within this context. When being sent during exposures executing only one integration (e.g. DET.NDIT = 1) the command has no effect.		
EXEC	Execute controller command (ASCII string). This is a low-level interface to all non-protected NGC controller electronics functions. To access also the protected functions the command "*NGC*" must be used.		
EXIT	Exit server. *NGCIRSW* will go to *OFF* state. The command aborts all current actions including running exposures. It is up to the calling application to check the exposure state in case running exposures must not be interrupted.		

EXPDRV (see also section 9.10.2)	**-list**	String	Return either a list of loaded drivers (*-list drv*) or a list of the dynamically loaded driver modules *(-list mod*). The names are separated by "newline".
	-load	String	Load external exposure driver module. The driver must exist as a shared library with the given string as library name ("*lib<string>.so*"). The library must be installed somewhere in the shared library search path (*$LD_LIBRARY_PATH*).
	-unload	String	Unload external exposure driver module.
	-select	String	Select and install an exposure driver by name.
	-fb	String	Select a feed-back channel where the progress messages are written to. Can be one of "*none*", "*mon*", "*stdout*", "*stderr*" or "*log*".
	-delete	Logical	Uninstall the exposure driver.
	-config	String	Pairs of "keyword" "value". Can be repeated to set multiple configuration keywords.
	-start	Logical	Start exposure sequence.
	-abort	Logical	Abort exposure sequence.
FRAME	**-module**	Integer	Module Id (starts with 1). A zero value refers to all modules.
	-name	String	Frame type name (mandatory).
	-gen	String	Generate frame (T|F).
	-store	String	Store frame to disk (T|F).
	-break	Integer	Number of frames of that type to be produced until the exposure can terminate (break condition).
	-local	String	Store frame data to local media on the NGC-LLCU (T|F).
	-win	Integer	4 window parameters (START-X, START-Y, NX, NY) defining a software window to be applied for that frame type on the given module(s).
KILL	Send a KILL signal to the server (SIGUSR2). *NGCIRSW* will go to *OFF* state.		

MERGE (see also section 10.5.1)	**-name**	String	Exposure base name or FITS-file name to be completed. When no explicit id is requested then a new merger id will be returned.
	-id	Integer	Explicitly request a merger id (1-4). When not specified the next available merger task will be used.
	-abort	Logical	Issue abort request to stop the merger with the given id. This requires an explicit id to be specified.
	-cleanup	Logical	Cleanup temporary files.
	-display	String	Indicates whether or not the merger process should launch a status display. The default value for this option is "T". When the option is not present then the value of the DET.MERGE.DISPLAY parameter is used.
	-resume	Logical	Resume an interrupted auto-merging procedure.
	-clear	Logical	Clear auto-merger list.
MSGDLOG	Disable auto-logging of messages sent or received by the application.		
MSGELOG	Enable auto-logging of messages sent or received by the application.		
NGC	Execute controller command (ASCII string). This is a low-level interface to all NGC controller electronics functions. The command gives access to both protected and non-protected instructions and has to be used with care. This is a TEST_COMMAND and is available only to developers!		
OFF	Close all devices and make all sub-processes terminate (*NGCIRSW* will go to *LOADED*-state).		
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 (*NGCIRSW* goes to *ONLINE* state). The voltage telemetry of all CLDC instances is automatically checked before.		
PAUSE	**-expoId**	Integer	Ignored.
	-at	String	Defines a pause 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 paused immediately.
PING	Make a check of the functioning of the server and send back an overall status message.		
RESET	Reset controller front-end.		
SELFTEST	**-function**	String	Execute a self-test (controller electronics and software) of the specified function(s). Valid functions are SEQ, CLDC, ADC, SHUTTER and ACQ.
	-repeat	Integer	Specifies, how often the given test is repeated in a loop.

SEQ	**-module**	Integer	Module Id (starts with 1). A zero value refers to all modules.
	-save	String	Save current clock-pattern configuration to a file. Unless a full path name is given, the file is stored at the default directory in $INS_ROOT.
	-tmo	Integer	Two values defining the timeout (in seconds) for the wait instruction (1st value) and a polling interval (2nd value). The default polling interval is 100 ms in case only the first parameter is set.
	-wait	String	Wait for the specified sequencer program event ("trigger" | "break" | "end").
	-stop	Logical	Stop sequencer and all associated acquisition processes.
	-start	Logical	Start sequencer and all associated acquisition processes.
	-pause	Logical	Pause sequencer program execution.
	-cont	Logical	Resume sequencer program execution.
	-trigger	Logical	Send software trigger.
SETUP	**-expoId**	Integer	Ignored.
	-file	String	Load setup from a file. Unless an absolute path is given, the setup file is searched in the directory: `$INS_ROOT/$INS_USER/COMMON/SETUPFILES/DET`
	-function	String	Pairs of "keyword" "value". Can be repeated to set multiple setup keywords.
	-default	Logical	Apply default setup as given in the currently loaded parameter default setup files (.dsup).
SIMULAT	Switch to simulation mode. A –function parameter (string) can be specified. If no function is specified or function is set to "HW", then only the controller electronics is simulated. If function is "LCU" then the controller electronics is simulated and all processes will be launched on the local host and the acquisition processes will start in non real-time mode (no buffer overrun check). Other values for the –function parameter may be used to put additional sub-system into simulation mode.		
STANDBY	Bring *NGCIRSW* to ***STANDBY***-state. The driver interface process(es) are started locally on the NGC-LLCU.		
START	**-expoId**	Integer	Passed through to DET.EXP.ID in the FITS-header(s) of the generated file(s).
	-at	String	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.
STATUS	**-expoId**	Integer	Ignored.
	-function	String	Retrieve the status of all given keywords. Can be repeated. If no function is given then the system operational state is queried.
STOPSIM	Switch from any simulation mode to normal mode.		

VERBOSE	**-on**	Logical	Switch generation of verbose messages on.
	-off	Logical	Switch generation of verbose messages off.
VERSION	Returns the actual server version (ASCII-string).		
WAIT	Wait for exposure to complete. The command immediately returns an intermediate reply indicating the current exposure status. The last reply is sent, when the exposure has finished. The –expoId parameter (integer) may be given, but is ignored.		

Table 6 Command Interface

Changes with respect to IRACE:

The command definition is backwards compatible. All IRACE commands for normal operation are still supported. Changes in command parameters mainly affect the SETUP/STATUS command. Details are given in sections 7 and 7.6. The MISC command is no longer supported.

5. Multiple Instances of DCS

Several instances of the control server (***ngcircon***) may run within the same environment. In this case an instance label (string) must be passed via the "*-inst*" command line option (see section 3.2) of the control server to distinguish between the systems. The instance label is used as appendix "***_<label>***" for both the database branch (see section 6) and for the server process name registered with the CCS environment (i.e. commands have to be sent to ***ngcircon_<label>***). The label must not exceed a length of 6 characters. The control server will exit with an error in case a process with the same label is already registered with the CCS environment.

All instances of *NGCIRSW* run fully independent. Synchronization can be done using external trigger signals (see section 11). If no high accuracy is needed, the synchronization can be also done at command interface level (e.g. issue an exposure start command at the proper time or use the command "*START –at <start-time>*" as described in sections 4 and 9.2).

Changes with respect to IRACE:

The instantiating scheme is the same as for IRACE, when restricting to decimal numbers for the instance label.

6. Database Interface

The file ***ngcircon.db*** contains the database branch definition for the control server. This file has to be included in the ***DATABASE.db*** file of the CCS environment. The following macros can be defined before each inclusion:

```
#define ngcirconINSTANCE ngcircon_myInstance
#define ngcirconROOT :Appl_data:…:myPoint
#define ngcirconRTD ngcrtdGui_myInstance
#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, the default value for ***ngcirconINSTANCE*** is "***ngcircon***" (which is used by the server when setting no instance label). ***ngcirconROOT*** defines the absolute path of the database root point. If no explicit root point is defined, then ***ngcirconINSTANCE*** is used as root point for the application.

The basic structure of the database is as follows:

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

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

The display branch will only be created when the ***ngcrtdRTD*** macro is defined (see section 14.4). When no ***ngcirconINSTANCE*** is defined (i.e. default instance) but still a display branch should be created then ***ngcirconRTD*** should be set to just "***ngcrtdGui***".

Database classes are provided for all sub-branches as described in the following sections. The classes are part of the ***ngcdcs*** SW module and are also used by the general purpose control server (see section 25 for details).

6.1. Control System Database Class

Attribute	Type	Description
serverName	BYTES64	Control server process name.
serverPid	INT32	Control server process id. A zero process id indicates that no server is running.
version	BYTES256	Version string.
detIndex	INT32	Detector category index (DETi).
sysCfgFile	BYTES256	Current system configuration file.
detCfgFile	BYTES256	Current detector configuration file.
opMode	BYTES32	DFE operation mode ("*NORMAL*", "*HW-SIM*", "*LCU-SIM*").
stateName	BYTES32	System state ("*OFF*", "*LOADED*", "*STANDBY*", "*ONLINE*").
state	INT32	System state (1 = "*OFF*", 2 = "*LOADED*", 3 = "*STANDBY*", 4 = "*ONLINE*").
subStateName	BYTES32	System sub-state ("*idle*", "*busy*", "*active*", "*error*").
subState	INT32	Sub-state (1 = "*idle*", 2 = "*busy*", 6 = "*active*", 8 = "*error*").
alarm	BYTES256	Alarm message.
numCldcMod	INT32	Number of CLDC modules.
numSeqMod	INT32	Number of sequencer modules.
numAdcMod	INT32	Number of ADC modules.
numAcqMod	INT32	Number of acquisition modules.
numPreampMod	INT32	Number of preamplifier interface modules.
fitsHdrSize	INT32	Number of FITS header blocks to reserve for merging.
polling	INT32	Status polling flag.
pollInterval	INT32	Polling interval in milliseconds (default = 1000 ms).
verboseLevel	INT32	Verbose output level.
logLevel	INT32	Logging level.
xterm	INT32	Start processes in separate x-terminal (0 = "off", 1 = "on").
autoOnline	INT32	Go to ONLINE state at startup.
autoStart	INT32	Start sequencer(s) when going to ONLINE state.
dic	BYTES256	Dictionary command line option. Format: "*-ld <dic1> -ld <dic2> …*"
currentAction	BYTE256	Action log.
param	Table	128 dynamic parameters (BYTES32 name, BYTES32 value).

Table 7 System Database Class (ngcdcsSYSTEM.class)

6.2. CLDC Module Database Class

Attribute	Type	Description		
name	BYTES64	CLDC module name.		
cfgFile	BYTES256	Voltage configuration file.		
statusName	BYTES32	CLDC status ("*enabled*" or "*disabled*").		
status	INT32	CLDC status (1 = "*enabled*", 0 = "*disabled*").		
clkMon1	INT32	Clock-line on analog clock-monitor 1.		
clkMon2	INT32	Clock-line on analog clock-monitor 2.		
preampOffset	DOUBLE	Preamplifier offset (Volt).		
diodeBias	DOUBLE	Diode bias (Volt).		
ilck	INT32	Interlock-enabled indicator.		
clk	Table (18 entries)	**nameLow**	BYTES64	Name of clock low level.
		nameHigh	BYTES64	Name of clock high level.
		voltageLow	DOUBLE	Voltage value of clock low.
		voltageHigh	DOUBLE	Voltage value of clock high
		range1Low	DOUBLE	Voltage range (minimum) for clock low level.
		range1High	DOUBLE	Voltage range (minimum) for clock high level.
		range2Low	DOUBLE	Voltage range (maximum) for clock low level.
		range2High	DOUBLE	Voltage range (maximum) for clock high level.
		connected	INT32	Clock is connected (0|1).
		reserved	INT32	To fill up 8 bytes boundary for the table entry.
dc	Table (28 entries)	**name**	BYTE64	Name of bias voltage.
		voltage	DOUBLE	Voltage value.
		range1	DOUBLE	Voltage range (minimum).
		range2	DOUBLE	Voltage range (maximum).
		connected	INT32	Bias channel is connected (0|1).
		reserved	INT32	To fill up 8 bytes boundary for the table entry.

Table 8 CLDC-Module Database Class (ngcdcsCLDC.class)

6.3. ADC Module Database Class

Attribute	Type	Description
name	BYTES64	ADC module name.
num	INT32	Total number of ADC units on board.
bitPix	INT32	Number of bits per pixel (16, 18, …).
enable	INT32	Number of enabled ADC units on board.
delay	INT32	Conversion strobe delay (in ticks).
packetSize	INT32	Data packet size (in 32-bit words).
packetCnt	INT32	Packet routing length. This is the number of data packets from the down-link the ADC module has to wait for before it transmits its own data.
convert1	INT32	Conversion strobe-1 enabled/disabled (0|1).
convert2	INT32	Conversion strobe-2 enabled/disabled (0|1).
opMode	INT32	Operational mode.
simMode	INT32	Simulation mode.
monitor1	INT32	Video-monitor 1.
monitor2	INT32	Video-monitor 2.
offset	DOUBLE	ADC offset (Volt).
clamp	INT32	Clamp.
filter	INT32	On-board filter (0 = 0.5 us, 1= 5 us)
ppena	INT32	Pre-processor enabled.
ppnx	INT32	Pre-processor x-dimension.
ppny	INT32	Pre-processor y-dimension.
ppnint	INT32	Pre-processor integration frame counter.
ppnsamp	INT32	Number of pre-processor sub-pixel samples.
ppFowler	INT32	Fowler mode.
ppInvert	INT32	Invert Fowler pairs.

Table 9 ADC-Module Database Class (ngcdcsADC.class)

6.4. Sequencer Module Database Class

Attribute	Type	Description
name	BYTES64	Sequencer module name.
clkFile	BYTES256	Clock pattern configuration file.
prgFile	BYTES256	Sequencer program file.
statusName	BYTES32	Sequencer status ("*running*", "*idle*", "*waiting*", "*failure*").
status	INT32	Sequencer status (0 = "*idle*", 1 = "*waiting*", 2 = "*running*", 4 = "*paused*", 0x1000000 = "*failure*").
timeFactor	INT32	Global dwell-time factor.
timeAdd	INT32	Global value to be added to dwell-time.
continuous	INT32	Continuous mode (0|1).
triggerOn	INT32	Trigger-enable flag (0|1).
triggerMode	INT32	Trigger mode (0 = "*direct*", 1 = "*by shutter*").
runCtrl	INT32	Run-control flag (0|1). When set to 1 the sequencer is automatically started via the external run-control line (i.e. starts synchronously with other sequencers in the system) and also forwards its own run-signal to this external line.
cvtExt	INT32	External convert flag (0|1). When set to 1 the external convert lines to/from the backplane are enabled.
startX	INT32	Read-out window (lower left x).
startY	INT32	Read-out window (lower left y).
nx	INT32	Read-out window (x-dimension).
ny	INT32	Read-out window (y-dimension).
clkMon1	INT32	Clock-line on digital clock-monitor 1.
clkMon2	INT32	Clock-line on digital clock-monitor 2.
dit	DOUBLE	Detector integration time.
minDit	DOUBLE	Minimum detector integration time.
tim	DOUBLE	Minimum TIM period.

Table 10 Sequencer-Module Database Class (ngcdcsSEQ.class)

6.5. Acquisition Module Database Class

Attribute	Type	Description		
name	BYTES64	Acquisition module name.		
statusName	BYTES32	Acquisition module status ("idle", "ready", "running", "failure", "*paused*").		
status	INT32	Acquisition module status (0 = "idle", 1 = "ready", 2 = "running", 4 = "failure", 8 = "*paused*").		
procName	BYTES256	Acquisition process name.		
burst	INT32	Number of bursts.		
burstSkip	INT32	Number of bursts to skip before starting data recording.		
continuous	INT32	Continuous mode (0|1).		
transfer	INT32	Sustained data transfer enabled/disabled (0|1).		
guiding	INT32	Secondary auto-guiding enabled/disabled (0|1).		
queueSize	INT32	Queue size (number of frames in queue). A value greater than zero enables the queue mode.		
diskMem	INT32	Disk memory extension enabled/disabled (0|1).		
frame	Table (32 entries)	**name**	BYTES64	Frame type name.
		gen	INT32	Generate frame.
		store	INT32	Store frame to disk.
		breakCond	INT32	Break condition for exposure.
		sx	INT32	Software window (lower left x)
		sy	INT32	Software window (lower left y)
		nx	INT32	Software window (x-dimension)
		ny	INT32	Software window (y-dimension)
host	BYTES64	Host name where acquisition process is launched (NGC-LLCU).		
cmdPort	INT32	Command port.		
dataPort	INT32	Data port.		
startX	INT32	Acquisition window (lower left x).		
startY	INT32	Acquisition window (lower left y).		
nx	INT32	Acquisition window (x-dimension).		
ny	INT32	Acquisition window (y-dimension).		

Table 11 Acquisition-Module Database Class (ngcdcsACQ.class)

6.6. Exposure Database Class

Attribute	Type	Description
expStatus	UINT32	Exposure status.
expStatusName	BYTES32	Exposure status name.
dataPath	BYTES256	Path to the directory where detector data is stored.
fileEvent	BYTES256	Intermediate file event (the file still has to be merged with the corresponding image data).
newDataFileName	BYTES256	New data file.
baseName	BYTES256	Exposure file base name (see sections 9.3 and 9.4).
naming	INT32	Naming scheme code (see section 9.4).
oneFile	INT32	Write all images to one file (0|1).
format	INT32	Data file format code (see section 9.3).
time	INT32	Estimation value for exposure time (in seconds).
countDown	INT32	Countdown. The countdown starts with the exposure time estimation value and decrements every second until the exposure has completed.
extFits	INT32	Generate extended FITS header.
drvStatus	UINT32	Exposure sequence driver status (see section 9.10).
drvStatusName	BYTES32	Exposure sequence driver status name (see section 9.10).

Table 12 Exposure Database Class (ngcdcsEXP.class)

6.7. Read-Out Mode Database Class

Attribute	Type	Description		
readModeName	BYTES64	Read-out mode name.		
readModeId	INT32	Read-out mode id.		
readModeList	Table (32 entries)	**name**	BYTES64	Read-out mode name.
		id	INT32	Read-out mode id.
		descr	BYTES256	Description.

Table 13 Read-Mode Database Class (ngcdcsMODE.class)

6.8. Guiding Mode Database Class

Attribute	Type	Description
offset	Vector[3] (FLOAT)	3 Entries: offset-x, offset-y and a quality value. The quality value indicates whether offset computation was successful (1.0, "good"), not successful (-1.0, "bad") or not even done (0.0, "unknown").

Table 14 Guiding-Mode Database Class (ngcdcs2AG.class)

6.9. Preamplifier Interface Database Class

Attribute	Type	Description
name	BYTES64	Preamplifier interface module name.
numChan	INT32	Number of channels in use (1-8).
monitor	INT32	Monitor channel.
gain	Vector[8] (INT32)	Gain (max. 8 channels).
bw	Vector[8] (INT32)	Bandwidth (max. 8 channels).
shortIn	Vector[8] (INT32)	Short input (0 or 1, max. 8 channels).
attn	Vector[8] (INT32)	Attenuation (0 or 1, max. 8 channels).
vref	Vector[8] (DOUBLE)	Reference voltages (max. 8 channels).

Table 15 Preamplifier Database Class (ngcdcsPREAMP.class)

6.10. Chopper Database Class

Attribute	Type	Description
state	INT32	Chopping mode on or off (0 or 1).
freq	DOUBLE	Chopping frequency (in Hz).
transtime	DOUBLE	Chopping phase transition time (in seconds).
tim	DOUBLE	Requested TIM period (in seconds).

Table 16 Chopper Database Class (ngcdcsCHOPPER.class)

7. Setup Command

This section shows a selection of the most important keywords. The setup keywords for the shutter-control (***DET.SHUTi.<name>***) are described in their dedicated section (section 20).

A complete list of all available keywords is given in the ***ESO-VLT-DIC.NGCDCS*** and ***ESO-VLT-DIC.NGCCON*** dictionaries which are archived in the ***dicNGC*** software module.

7.1. Exposure Setup

Keyword	Type	Description
DET.FRAM.EXTHDR	**Logical**	Define whether or not the extended FITS-header including all voltage telemetry keywords is enabled. This keyword should normally be set to 'T' (which is also the default value).
DET.FRAM.FILENAME	**String**	Defines the (base-) filename for the data file(s) produced during the next exposure. The actual file name still depends on the ***DET.FRAM.FORMAT*** and ***DET.FRAM.MULTFILE*** keywords as described in section 9.3.
DET.FRAM.FORMAT	**String**	Defines the FITS file format to be used ("*single*", "*cube*", "*extension*", "*single-ext*", "*cube-ext*") as described in section 9.3.
DET.FRAM.MULTFILE	**Logical**	Defines whether it is allowed to store multiple files during one exposure or not. The parameter only takes effect in case the "*extension*" format is selected. If set to 'F' then only one file containing all data in image extensions will be created. See section 9.3 for details.
DET.FRAM.SPLIT	**Logical**	This keyword instructs the application to split cubes of glued images into extensions per chip. The keyword only has an effect when the file format is "*cube-ext*" (see section 9.3). There might be a considerable performance drawback when the keyword is set to 'T'. Default value is 'F'.
DET.FRAM.NAMING	**String**	Defines the frame naming scheme ("*request*", "*sequence*" or "*auto*") as described in section 9.4.
DET.EXP.DRVNAME	**String**	Select exposure driver by its name (see section 9.10.2).
DET.EXP.DRVFB	**String**	Exposure driver feed-back channel (see section 9.10.2). Valid values are "*none*", "*mon*", "*stdout*", "*stderr*" or "*log*".

Table 17 Exposure Setup Keywords

7.2. ADC Module Setup

Keyword	Type	Description
DET.ADCi.CLAMP	Logical	Select "*clamp-and-sample*" instead of "*filter*".
DET.ADCi.FILTER	Integer	Selects the filter on the board: 0 = 0.5us, 1 = 5us
DET.ADCi.DELAY	Integer	Conversion strobe delay on ADC-module (in ticks).
DET.ADCi.OFFSET	Double	ADC offset voltage.
DET.ADCi.MONi	Integer	Selects the video channel to be monitored on video monitor 1 or 2 (monitor number 2 is not supported by the current hardware revisions).
DET.ADCi.ENABLE	Integer	Number of ADC units to enable on an ADC module. It can be set to zero to disable the whole module.
DET.ADCi.PKTSIZE	Integer	Data packet size (in 32-bit words). For 16-bit ADCs this must be a multiple of half the number of enabled ADC units on that module.
DET.ADCi.PKTCNT	Integer	Packet routing length. This is the number of data packets from the down-link the ADC module has to wait for before it transmits its own data.
DET.ADCi.FIRST	Logical	Designate module as first ADC module in chain.
DET.ADCi.CONVERTi	Logical	Enable conversion on convert-strobe 1 and/or 2. The last index specifies the strobe number (1 or 2).
DET.ADCi.OPMODE	Integer	Select ADC operational mode (0 = normal, 1 = simulation)
DET.ADCi.SIMMODE	Integer	Select ADC simulation mode (0 = channel numbers, 1 = counter). Only applied when DET.ADCi.OPMODE is set to 1 (simulation).
DET.ADCi.DLINK	Logical	Enable/disable dual-link mode.
DET.ADCi.PPENA	Logical	Enable/disable the ADC prep-processor (see section 24.4).
DET.ADCi.PPNINT	Integer	Pre-processor integration frame counter (see section 24.4).
DET.ADCi.PPNSAMP	Integer	Number of pre-processor sub-pixel samples (see section 24.4).
DET.ADCi.PPNX/NY	Integer	Pre-processor frame dimension (see section 24.4).
DET.ADCi.PPFOWLER	Logical	Enable/disable Fowler mode (see section 24.4).
DET.ADCi.PPINVERT	Logical	Invert Fowler pairs (see section 24.4).

Table 18 ADC Module Setup Keywords

7.3. Voltage Setup

Keyword	Type	Description
DET.CLDCi.FILE	String	Load a new voltage configuration file.
DET.CLDCi.INITFILE	String	Additional voltage configuration which is loaded once the system is enabled before applying the final voltage configuration.
DET.CLDCi.DCGNi	Double	Gain factor for bias voltages: *V_dac = V_set / gain*. *V_dac* is the voltage programmed in the DAC and *V_set* is the value entered by the user. When the last index is not given then the gain is applied to all bias voltages.
DET.CLDCi.CLKGNi	Double	Gain factor for clock voltages: *V_dac = V_set / gain*. *V_dac* is the voltage programmed in the DAC and *V_set* is the value entered by the user. When the last index is not given then the gain is applied to all *clock-low* and *–high* voltages.
DET.CLDCi.CLKLOGNi DET.CLDCi.CLKHIGNi	Double	Gain factor for clock low/high voltages: *V_dac = V_set / gain*. *V_dac* is the voltage programmed in the DAC and *V_set* is the value entered by the user. When the last index is not given then the gain is applied to all *clock-low/high* voltages.
DET.CLDCi.TELDCGN	Double	Telemetry gain for the bias voltages: *V_tel = V_adc / gain* *V_tel* is the value shown to the user and *V_adc* is the value measured with the telemetry ADC.
DET.CLDCi.TELCLKGN	Double	Telemetry gain for the clock voltages: *V_tel = V_adc / gain* *V_tel* is the value shown to the user and *V_adc* is the value measured by the telemetry ADC.
DET.CLDCi.DCi	Double	Voltage for the given bias channel. The range will be checked before the new value is applied.
DET.CLDCi.CLKLOi DET.CLDCI.CLKHIi	Double	Voltage for the low/high level of the given clock. The range will be checked before the new value is applied.
DET.CLDCi.DCXi	Integer	Register voltage for the given bias channel. The keyword is interpreted as hexadecimal value. The range will be checked before the new value is applied.
DET.CLDCi.CLKLOXi DET.CLDCi.CLKHIXi	Integer	Register value for the low/high level of the given clock. The keyword is interpreted as hexadecimal value. The range will be checked before the new value is applied.
DET.CLDCi.DCERRi	Double	Error compensation for bias voltage. The value is subtracted from the setup value up to a limit as given by the DET.CLDC.CALMAX keyword.
DET.CLDCi.CLKLOERRi DET.CLDCI.CLKHIERi	Double	Error compensation for the given clock. The value is subtracted from the setup value up to a limit as given by the DET.CLDC.CALMAX keyword.
DET.CLDCi.PREAMP	Double	Preamplifier offset voltage.
DET.CLDCi.DIODE	Double	Diode bias voltage.
DET.CLDCi.MONi	Integer	Selects the clock to be monitored on analog clock-monitor 1 or 2.

Table 19 Voltage Setup Keywords

7.4. Sequencer Setup

Keyword	Type	Description
DET.SEQ.BREAK	Logical	When set to 'T' the sequencer will always run till next break-point when the "*SEQ –stop*" command is issued. Otherwise the sequencer will stop immediately when the "*SEQ –stop*" command is issued (see section 16.3).
DET.SEQ.BREAKTMO	Integer	Timeout (in seconds) for reaching the next break-point when the "*SEQ –stop*" command is issued. The default timeout for reaching the next break-point is 10 seconds. A negative value indicates an infinite timeout (not recommended). A zero value lets the system choose. The zero value may cause additional overheads in order to compute the maximum time between two break-points. Also the system may then still switch to "immediate stop" when no proper timeout is found (e.g. no break-point within an infinite loop). See section 16.3.
DET.SEQi.CLKFILE	String	Load a new clock-pattern definition file.
DET.SEQi.PRGFILE	String	Load a new sequencer program file.
DET.SEQi.DIT	Double	Detector integration time to be set for the sequencer instance given in the SEQ index.
DET.SEQi.CONT	Logical	Defines whether the sequencer program should run continuously or whether it should be re-started whenever a new exposure is started. The continuous mode introduces a (worst case) overhead of one detector integration but prevents residual effects e.g. due to detector saturation during re-start.
DET.SEQi.TIMEFAC	Integer	Global dwell-time factor for clock-pattern state length. Can be used to slow down the overall detector read-out. The factor is only applied to states having the modification flag set in the clock pattern definition (see section 16.1).
DET.SEQi.TIMEADD	Integer	Global value (in ticks of 10 ns) to be added to the dwell time of each clock pattern state. The value is only applied to states having the modification flag set in the clock pattern definition (see section 16.1).
DET.SEQi.TRIGGER	Logical	Enable/disable trigger mode.
DET.SEQi.TRIGMODE	Integer	Sequencer trigger mode. Valid values are 0 (direct trigger) or 1 (shutter). Other numbers may apply for special applications. The default value is 0.
DET.SEQi.TIMAUTO	Logical	Enable/disable the automatic computation of the minimum TIM-period.
DET.SEQi.INITPAT	String	Default initial clock pattern state vector to be applied after sequencer program reset. The format is a binary string with max. 64 characters (0|1).
DET.SEQi.INITLOW	Integer	Default initial clock pattern low word to be applied after sequencer program reset. The parameter is interpreted as hexadecimal value.
DET.SEQi.INITHIGH	Integer	Default initial clock pattern high word to be applied after sequencer program reset. The parameter is interpreted as hexadecimal value.
DET.SEQi.PATi.CLKi	String	Set a new state vector on a certain clock in a certain pattern. This overwrites the vector defined by the clock-pattern definition file.

Keyword	Type	Description
DET.SEQi.MONi	Integer	Selects the clock to be monitored on digital clock-monitor 1 or 2.
DET.SEQi.CVTEXT	Logical	Enable/disable external convert. When set to 'T' the external convert lines to/from the backplane are enabled.
DET.SEQi.RUNCTRL	Logical	Enable/disable Run-control. When set to 'T' the sequencer is automatically started via the external run-control line (i.e. starts synchronously with other sequencers in the system) and also forwards its own run-signal to this external line.
DET.SEQi.TIMESIM	Logical	Enable/disable sequence time simulation. When the system is not in simulation mode then the keyword has no effect. The default value is 'T'.

Table 20 Sequencer Setup Keywords

7.5. Data Acquisition Setup

Keyword	Type	Description
DET.ACQi.CONT	Logical	Can be set to T to avoid the (worst case) overhead of one detector integration, when a new exposure is started and the associated sequencer is running in continuous mode. Only applicable to consecutive exposures where there is no change in the field of view (no telescope movement, no filter change, etc.).
DET.ACQi.TRANSFER	Logical	Can be set to T to establish a sustained data transfer from the acquisition process to the data transfer task of the control server. The data transfer task will continuously request data from the acquisition process and (if applicable) do some application specific post-processing. The keyword is used for control loops such as secondary auto-guiding.
DET.ACQi.GUIDING	Logical	Enable/disable the guiding mode on the given acquisition process. When enabled the offset correction vector for secondary auto-guiding is continuously transferred from the acquisition process to the database. This does not necessarily require the other frames to be transferred continuously as controlled with the ***DET.ACQi.TRANSFER*** keyword.
DET.ACQi.PROC	String	The keyword sets a new acquisition process by name. This overrides the name defined in the read-out mode configuration.
DET.ACQi.CHAN	Integer	The keyword selects the DMA channels in use. The 32-bits indicate up to 32 enabled data channels. If the keyword has a negative value (= system default) then the channel selection is disabled and the device driver default value is used.
DET.ACQi.BURST.NUM DET.ACQi.BURST.SKIP	Integer	Parameters for the burst-mode as described in section 9.10.
DET.ACQi.BURST.PROC	String	Name of the acquisition process to be used when the burst mode is selected (default is "*ngcppBurst*").
DET.ACQi.QUEUE	Integer	Size of internal frame queue (number of frames). The queue mode is enabled when the queue size is larger than zero.
DET.ACQi.QDUMP	Logical	Dump queue data to local disk. When set to T the queue data is not transferred to the control server but stored to the local disk instead. The control server only creates a header file containing a reference to the data.
DET.ACQi.LPATH	String	Path to local storage media.
DET.ACQi.DISKMEM	Logical	Enable/disable disk-memory extension. When enabled then in queue-mode and in burst mode the data is buffered on some pre-allocated disk-space. This reduces performance but provides unlimited buffering capacity.
DET.ACQi.DMSIZE	Integer	Define the size (in Mbytes) of the disk space to be allocated for the disk memory extension. A zero size frees the disk space.
DET.ACQi.DMPATH	String	Root directory of the disk memory extension.
DET.ACQi.CONNECT	Logical	Enable/disable the data connection to the acquisition process. By default the data connection is always enabled. When disabled no transfer, no guiding and no exposure can be done.
DET.ACQi.EXP	Logical	Enable/disable the exposures on the acquisition module. By default the exposures are always enabled. When disabled no exposure data will be received from the acquisition process.

Keyword	Type	Description
DET.READ.CURNAME	String	Selects the current read-out mode by name. The read-out modes are defined in the detector configuration file. A list of available read-out modes is also stored in the online database attribute (***<alias>ngcircon:mode.readModeList,*** see section 6.7) or can be retrieved via the ***STATUS*** command (*DET.READ.AVAIL*).
DET.READ.CURID	Integer	Selects the current read-out mode by its unique id.

Table 21 Data Acquisition Setup Keywords

7.6. Preamplifier Setup

Keyword	Type	Description
DET.PREAMPi.NUMCHAN	Integer	Number of channels.
DET.PREAMPi.MON	Integer	Monitor channel.
DET.PREAMPi.GAINi	Integer	Gain (per channel).
DET.PREAMPi.BWi	Integer	Bandwidth (per channel).
DET.PREAMPi.SHORTi	Logical	Short input (per channel).
DET.PREAMPi.ATTNi	Integer	Attenuation (per channel).
DET.PREAMPi.VREFi	Double	Reference voltage (per channel).
DET.PREAMPi.INVERT	Logical	Invert signals.
DET.PREAMPi.VREFCMD	Integer	Preamplifier reference voltage DAC control command. This is a 4-bit integer value to setup the DAC. The control command is common to all reference voltages of one preamplifier module.
DET.PREAMPi.BUSCMD	Integer	I2C-bus control command. This is a 16-bit integer value to setup the I2C-bus. It goes directly into the upper 16 bits of the I2C port address. The control command is common to all register accesses through the I2C-bus of one preamplifier module.

Table 22 Preamplifier Setup Keywords

8. Status Command

This section shows a selection of the most important keywords. Additionally the value of all of the ***SETUP*** keywords described in section 7 can be retrieved via the ***STATUS*** command. The status keywords for the shutter-control (***DET.SHUTi.<name>***) are described in their dedicated section (section 20).

A complete list of all available keywords is given in the ***ESO-VLT-DIC.NGCDCS*** and ***ESO-VLT-DIC.NGCCON*** dictionaries which are archived in the ***dicNGC*** software module.

8.1. System Status

Keyword	Type	Description
DET.CON.OPMODE	String	Returns the current operational mode of the detector front-end system ("*NORMAL*", "*HW-SIM*" or "*LCU-SIM*").
DET.EXP.STATUS	String	Actual exposure status.
DET.EXP.DRVNAME	String	Name of the currently installed exposure driver (see section 9.10.2). Can be an empty string when no driver is installed.
DET.EXP.DRVSTAT	String	Actual exposure driver state (see section 9.10.2). The state can be one of "*inactive*", "*processing*", "*success*", "*failure*" or "*aborted*". An empty string indicates that no driver is currently installed.
DET.EXP.DRVMSG	String	Contains a message when the exposure driver state changes to "*failure*" or "*aborted*".
DET.CHOP.FREQ	Double	Returns a maximum value for the chopping frequency to be used with the currently loaded sequencer program (see section 22).
DET.CHOP.TIM	Double	Returns a minimum TIM period to be used for synchronization (see section 22).

Table 23 System Status Keywords

8.2. ADC Module Status

Keyword	Type	Description
DET.ADCi.HWSTAT	Integer	Content of the ADC-module status register (in hexadecimal format).
DET.ADCi.SWSTAT	Integer	Software mirror of the ADC-module control register (in hexadecimal format).
DET.ADCi.TEMP	Double	Board temperature in decC.

Table 24 ADC Module Status Keywords

8.3. Voltage Status

Keyword	Type	Description
DET.CLDCi.STATUS	String	The actual CLDC status ("*enabled*" or "*disabled*").
DET.CLDCi.TEMP	Double	Board temperature in decC.
DET.CLDCi.ILCK	Logical	The keyword indicates whether the external interlock mechanism is enabled or disabled.
DET.CLDCi.DCTi	Double	Telemetry value of bias voltage.
DET.CLDCi.CLKLOTi DET.CLDCi.CLKHITi	Double	Telemetry value of clock low/high-level.

Table 25 Voltage Status Keywords

8.4. Sequencer Status

Keyword	Type	Description
DET.SEQi.STATUS	String	The keyword can be used to query the sequencer status. The status can be one of "idle", "running", "waiting" or "failure".
DET.SEQi.REGSTAT	Integer	Content of the sequencer status register (in hexadecimal format).
DET.SEQi.TEMP	Double	Board temperature in decC.
DET.SEQi.WAITING	Logical	The keyword can be used to query the sequencer "*waiting*" state.
DET.SEQi.MINDIT	Double	Minimum integration time (in seconds).
DET.SEQi.TIM	Double	Minimum TIM-period (in seconds) to be applied for triggering the sequencer program. A zero value indicates that the sequencer program does not contain any trigger-point. A negative value indicates that an error occurred when computing the trigger points.
DET.SEQi.EXPTIME	Double	Estimated value for the exposure duration (in seconds).
DET.SEQi.REALDIT	Double	Actual detector integration time value (in seconds).
DET.SEQi.LASTPAT	String	Last clock state vector executed by the currently loaded sequencer program. The format is a binary string with 64 characters (0|1).
DET.SEQi.FIRSTPAT	String	First clock state vector executed by the currently loaded sequencer program. The format is a binary string with 64 characters (0|1).
DET.SEQi.PATi.STATLOWi	Integer	Low RAM clock state of a pattern. The value is returned in hexadecimal format.
DET.SEQi.PATi.STATHIGHi	Integer	High RAM clock state of a pattern. The value is returned in hexadecimal format.

Table 26 Sequencer Status Keywords

8.5. Data Acquisition Status

Keyword	Type	Description
DET.ACQi.MAXOVHD	Double	Maximum value of the measured data transfer overhead (in seconds).
DET.ACQi.MEANOVHD	Double	Mean value of the measured data transfer overhead (in seconds).
DET.ACQi.DATAPORT	Integer	Actual data port number of the acquisition process.
DET.READ.AVAIL	String	Returns a list with all read-out modes currently defined. The format of the list is: "<*id*>:<*name*>|<*id*>:<*name*>|…"
DET.READ.FRAMES	String	Returns a list with all frame-types currently defined (see section 9.5).

Table 27 Data Acquisition Status Keywords

9. Exposure Handling

9.1. Description

When the detector system is ***ONLINE***, the data acquisition is usually running continuously and integrations are done one after another. Starting an exposure basically means to start data taking and to start transferring the data to a file on the IWS. When the exposure has started, the exposure status will be "***integrating***". When the header of the last file has been received (i.e. the break-conditions for all frames have been reached – see section 9.5) the exposure status goes to "***transferring***". When the last file has been stored on disk, the exposure status 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***".

Generally the field of view can already be changed (e.g. telescope can be moved) when the exposure status changes to "***transferring***" (all data for this exposure have been read-out). But in any case one of the completion states ("***success***", "***failure***", "***aborted***") must have been reached, before a new setup can be done or before a new exposure can be started (otherwise an error will be reported, stating that the exposure was still active). The exposure status is stored in the database attribute '***<alias>ngcircon:exposure.expStatus***'. It can take one of the following values:

```
1   - inactive
2   - pending
4   - integrating
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***'.

Changes with respect to IRACE:

The exposure states are backwards compatible. The database attribute name for the exposure status name has changed from "expStatusN" to "expStatusName").

9.2. Commands

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. 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 has 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 acquisition 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]>
```

This 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.

The exposure can be aborted using the ***ABORT*** command. In this case no data file is generated unless a frame was already received 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 status 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 status. A running exposure always has to be waited for completion before starting the next one or before issuing a new setup. The typical command sequence will be:

a) START – WAIT
b) START – END – WAIT
c) START – ABORT – WAIT

Alternatively the exposure status attribute in the database (see section 9.1) may be used to wait for a specific state (e.g. "***transferring***").

Changes with respect to IRACE:

The exposure control commands are backwards compatible. The DET.IRACE.SEQCONT SETUP keyword has changed to DET.SEQi.CONT and is set per sequencer instance.

9.3. 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 these overheads the frames may be stored into data-cubes (***DET.FRAM.FORMAT = "cube"***). One cube will 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. When the cube size exceeds a 2GB file size limit a new cube file will be opened for each 2GB chunk and an auto-incremented number is added to the cube file name (e.g. *<cube>_DIT_1.fits*, *<cube>_DIT_2.fits*, and so on). There is still the possibility to overcome the 2GB file size limit by making use of the 64-bit large file extension for 32-bit Linux operating systems (see section 10.3). This is enabled via the ***DET.FRAM.LRGFILE*** keyword which is then typically set in the system configuration file:

```
DET.FRAM.LRGFILE T; # large files enabled
DET.FRAM.LRGFILE F; # large files disabled (split cubes into chunks)
```

The "*large file*" extension has to be used with care as not all applications are able to deal with such data.

In some cases it is desired to have the data-cube in an image extension rather than directly after the primary header unit. This can be achieved without touching the data blocks by reshuffling the primary header and adding an extension header in the last header block(s). The ***ngcbCube2Ext*** tool (see section 27.5) converts a data-cube file into that format. Alternatively the cube extension format (***DET.FRAM.FORMAT*** = "***cube-ext***") can be selected which will instruct the control server to call this tool before issuing the file event. It is recommended to increase the number of reserved blocks (***DET.FRAM.NUMBLOCK*** – see section 9.7) by at least 4 in order to have enough space for the extension header.

In the case of multi-chip mosaics (see section 9.8) the data-cube format will by default store cubes of glued images. The data of a mosaic image are acquired together and cannot simply be split to file positions not yet known at acquisition time. If the data-cube for each chip should be stored in separate extensions then a file re-processing is required. The ***ngcbSplitCube*** tool (see section 27.6) can be used to split a data-cube into extensions "*per chip*" (offline file conversion). When "***cube-ext***" file format is selected then the frame splitting can be done automatically by setting the ***DET.FRAM.SPLIT*** parameter to '***T***' (see section 7.1).

Caution:
The file format "cube-ext" together with the frame splitting option can cause significant transfer overheads and must be used with care.

The standard file format is to store the frames produced during one exposure into binary image extensions (***DET.FRAM.FORMAT = "extension"***). If the data are coming from different acquisition processes they typically will 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 on each other before saving data to disk (***DET.FRAM.MULTFILE = "T"***). With the right configuration this improves 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 set 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 the loss of 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 whole exposure has finished. The "***single-ext***" file format may be chosen to allow such intermediate quick-looks. This requires a file merging to be done by the higher level SW (OS) before passing the data to the archive.

Changes with respect to IRACE:

The file formats "single" and "cube" are still supported. A cube now must contain only one frame type. The cube definitions for multiple types are obsolete and have been replaced with the full support of binary image extensions. The logical keyword defining to store data into cubes or not (DET.FILE.CUBE.ST) has been replaced with the more general DET.FRAM.FORMAT keyword. The mosaic handling as done with IRACE is still possible (store to single FILES and merge by higher level SW). The file format "single-ext" must be used in this case to instruct the control server to add the additional FITS header keywords needed for the image extension format. In future versions this format may become obsolete (TBD) so it should be used with care.

9.4. Naming Schemes

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 (***DET.FRAM.FILENAME***). 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) or when several data cubes per frame-type have to be created in case the 2GB Linux file size limit is exceeded (see section 9.39.3).

Changes with respect to IRACE:

The three naming schemes have already been used with IRACE. The file names when storing to single files have changed (frame-name, frame-number are swapped). Setup keywords have changed:

DET.EXP.NAMING.TYPE	->	*DET.FRAM.NAMING*
DET.EXP.NAME, DET.EXP.SEQ.NAME	->	*DET.FRAM.FILENAME*
DET.EXP.SEQ.NO	->	*DET.FRAM.SEQIDX*

9.5. 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. A zero value for the dimension (*nx*, *ny*) indicates that the full frame will be requested from the acquisition process.

Usually an exposure is finished, when the INT-frame (containing the mean value of several integrations) 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, which indicates the number of frames of that type, which 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>] [-local T|F] [-win sx sy nx ny]*"

The frame setup can be done "per process" by specifying an acquisition module id. When the module id is zero (or in case no module-id is passed), then the command will refer to "all" modules.

For each acquisition module the available frame types are stored as a table (with the fields: *name*, *generate*, *store*, *breakCond*, *sx*, *sy*, *nx*, *ny*) 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*". The list format is:

```
<Mod.-Nr>:name1 g s b [sx sy nx ny]|name2 g s b [sx sy nx ny]|…|nameN g s b
[sx sy nx ny]||<Mod.-Nr>:name1 g s b [sx sy nx ny]|…
```

Module numbers start with 1. <g> and <s> are 0 or 1 and give the **g**enerate/**s**tore flags. <b> is the **b**reak counter for the exposure termination condition. [sx sy nx ny] gives the software window to be applied for this frame.

Changes with respect to IRACE:

The FRAME command is backwards compatible with IRACE. The individual software window setting and the possibility to configure the frame-setup "per acquisition module" have been added. The database point for the frame-table has changed. The STATUS keyword for retrieving the frame list has changed from DET.NCORRS.FRAMES to DET.READ.FRAMES. The format of the returned list also has changed to include the information to which acquisition module the frame types belong.

9.6. Frame Parameters

The ***FRAME*** command was introduced to ensure that the actions "*gen*", "*store*", "*break*" and also the SW-window are always consistent with the respective frame type. However applications may prefer to avoid this additional "non-standard" command and use setup parameters instead. It is possible to assign a set of control keywords to each frame-type in the system configuration file (section 17.1):

```
DET.FRAM1.NAME "DIT";    # frame name
DET.FRAM1.ACQ  "1,2,3"; # acquisition process list

DET.FRAM2.NAME "STDEV"; # frame name
DET.FRAM2.ACQ  "1";     # (STDEV on process 1)

DET.FRAM3.NAME "STDEV"; # frame name
DET.FRAM3.ACQ  "2,3";   # (STDEV on process 2 and 3)
```

The frame control is then done via the respective control keywords:

```
DET.FRAMi.GEN   - Generate the frame
DET.FRAMi.STORE - Store the frame to disk during exposure
DET.FRAMi.BREAK - Exposure break counter
DET.FRAMi.LOCAL - Store the frame data to local media on NGC-LLCU
DET.FRAMi.STRX  - SW window lower left corner in x
DET.FRAMi.STRY  - SW window lower left corner in y
DET.FRAMi.NX    - SW window x-dimension
DET.FRAMi.NY    - SW window y-dimension
```

The keywords apply to all acquisition processes defined by the ***DET.FRAMi.ACQ*** keyword. A "*0*" value indicates that the frame control keywords refer to all acquisition processes (this is the default when the keyword is omitted).

Where no special frame setup is given in the system configuration file the following default assignment is used:

```
DET.FRAM1.NAME = "DIT"
DET.FRAM2.NAME = "INT"
DET.FRAM3.NAME = "STDEV"
DET.FRAM4.NAME = "INTERM-INT"
DET.FRAM5.NAME = "HCYCLE1"
DET.FRAM6.NAME = "HCYCLE2"
DET.FRAM7.NAME = "SAMPLE"
DET.FRAM8.NAME = "INTERM-DIT"
```

For large systems the numbered assignment scheme may become confusing. In those cases it is possible to declare a set of special keywords to be used for the frame control:

```
DET.FRAMi.PARAM = "DET.DIT"
```

The server will then automatically introduce for each declared type the new setup parameters:

```
<DET.FRAMi.PARAM>.GEN
<DET.FRAMi.PARAM>.STORE
<DET.FRAMi.PARAM>.BREAK
<DET.FRAMi.PARAM>.STRX
<DET.FRAMi.PARAM>.STRY
<DET.FRAMi.PARAM>.NX
<DET.FRAMi.PARAM>.NY
```

In the above example:

```
DET.DIT.GEN,
DET.DIT.STORE,
DET.DIT.BREAK,
DET.DIT.STRX/Y,
DET.DIT.NX/Y
```

The only thing to do for each application is to declare those generic parameters in a specific dictionary.

9.7. FITS-Header Contents

NGCIRSW provides the primary header filled with the exposure time stamps and a snapshot of the detector system keywords taken at exposure start time. This also includes the setup and telemetry values for all clock- and bias-voltages of all CLDC boards in use. The image extension headers inherit the values from the primary header. *NGCIRSW* fills in the actual dimension and data-type keywords and also provides the chip information as given in the detector configuration file. Image extensions containing bad-data (e.g. chip or associated ADC-channels are broken) are marked as such with the ***DET.CHIP.LIVE*** keyword, which is set to '***F***' in this case. The bad data are left in the extension data array for the purpose of further inspection. If in case of detector mosaics (see section 9.8) a chip is masked out (e.g. due to windowing), then a place holder (extension header with NAXIS1/2=0) will still be created in order to give the information, that a chip of that type is present but does not contribute any data-pixels to the FITS file.

The information, which system parameters (if used in the actual context) will appear in the FITS header, is defined in the dictionaries [AD13]. So it always has to be ensured that the right (e.g. latest) version of ***ESO-VLT-DIC.NGCDCS*** is installed in the ***$INS_ROOT/SYSTEM/Dictionary/*** directory.

The system reserves some space in the primary FITS-header for adding FITS keywords. By default 16 blocks are reserved (each block has 36 lines – so this is 576 keywords in total). The value can be overwritten with the ***DET.FRAM.NUMBLOCK*** keyword in the system configuration file – e.g.:

```
DET.FRAM.NUMBLOCK  24;  # default number of FITS-header blocks
```

The value can also be overwritten via ***SETUP*** command. Additionally there is a database attribute for this:

```
<alias>ngcircon:system.fitsHdrSize
```

The system will take the larger one of either the SETUP value or the database attribute.

It is possible to prevent individual controller modules from exporting their status to the FITS header. The following keywords can be set either at run-time or in the system configuration file (see section 17.1.2):

```
DET.SEQi.FITS   T|F; # FITS enabled
DET.CLDCi.FITS  T|F; # FITS enabled
DET.ADCi.FITS   T|F; # FITS enabled
```

When set to "***F***" then the status of the respective module is not exported to the FITS header. The default value of these keywords is "***T***".

Primary Header:

```
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    = '2006-09-13T08:14:10.4730' / Date the file was written
EXPTIME = 5.0000000 / Integration Time
ORIGFILE= 'ngc.fits' / Original File Name
MJD-OBS =       53991.34311830 / 2006-09-13T08:14:05.4214
DATE-OBS= '2006-09-13T08:14:05.4214' / Observing date
HIERARCH ESO DET FRAM UTC    = '2006-09-13T08:14:05.4156' / File Creation
Time
HIERARCH ESO DET EXP ID      = 0 / Unique exposure ID
HIERARCH ESO DET DID         = 'ESO-VLT-DIC.NGCDCS-1.38' / NGCDCS dictionary
HIERARCH ESO DET CON OPMODE  = 'HW-SIM' / Operational Mode
HIERARCH ESO DET READ CURID  = 2 / Used readout mode id
HIERARCH ESO DET READ CURNAME= 'Double' / Used readout mode name
HIERARCH ESO DET ACQ1 PROC   = 'ngcppTemplate' / Process name
HIERARCH ESO DET ACQ1 CONT   = F / Acq.-process continuous mode
HIERARCH ESO DET SEQ1 DIT    = 5.0000000 / Integration Time
HIERARCH ESO DET SEQ1 MINDIT = 0.3301660 / Minimum DIT
HIERARCH ESO DET NDIT        = 10 / # of Sub-Integrations
HIERARCH ESO DET SEQ1 CONT   = F / Continuous mode active
HIERARCH ESO DET SEQ1 RUNCTRL= T / Run-control active
HIERARCH ESO DET SEQ1 CVTEXT= T / External convert active
HIERARCH ESO DET SEQ1 TRIGGER= F / Trigger mode active
HIERARCH ESO DET SEQ1 TIMEFAC= 2 / Dwell-time factor
HIERARCH ESO DET SEQ1 TIMEADD= 0 / Dwell-time add value
HIERARCH ESO DET SEQ1 WIN STRX= 1 / Read-out window start-x
HIERARCH ESO DET SEQ1 WIN STRY= 1 / Read-out window start-y
HIERARCH ESO DET SEQ1 WIN NX = 1024 / Read-out window NX
HIERARCH ESO DET SEQ1 WIN NY = 1024 / Read-out window NY
HIERARCH ESO DET ADC1 OFFSET = 1.000 / Offset value for ADC (volt)
HIERARCH ESO DET ADC1 DELAY  = 0 / Conv.-strobe delay (ticks)
HIERARCH ESO DET ADC1 CLAMP  = F / Clamp-and-Sample
HIERARCH ESO DET ADC1 FILTER = 0 / Filter (0 = 0.5us, 1 = 5us)
HIERARCH ESO DET CLDC1 PREAMP= 1.000 / Preamplifier offset (volt)
HIERARCH ESO DET CLDC1 DIODE= 1.000 / Diode bias (volt)
HIERARCH ESO DET CLDC1 CLKOFF= 2.000 / Clock voltage offset (volt)
HIERARCH ESO DET CLDC1 DCOFF = 2.000 / Bias voltage offset (volt)
HIERARCH ESO DET CLDC1 CLKLO1= 0.000 / Setup value for clock low (volt)
HIERARCH ESO DET CLDC1 CLKLOT1= 0.001 / Telemetry value for clock low (volt)
HIERARCH ESO DET CLDC1 CLKHI1= 1.000 / Setup value for clock high (volt)
HIERARCH ESO DET CLDC1 CLKHIT1= 1.001 / Telemetry value for clock high
(volt)
 […]
HIERARCH ESO DET CLDC1 CLKLO18= 0.000 / Setup value for clock low (volt)
HIERARCH ESO DET CLDC1 CLKLOT18= 0.030 / Telemetry value for clock low
(volt)
HIERARCH ESO DET CLDC1 CLKHI18= 0.000 / Setup value for clock high (volt)
HIERARCH ESO DET CLDC1 CLKHIT16= 0.031 / Telemetry value for clock high
(volt)
HIERARCH ESO DET CLDC1 DC1   = 0.100 / Setup value for bias voltage (volt)
HIERARCH ESO DET CLDC1 DCT1  = 0.131 / Telemetry value for bias voltage
(volt)
 […]
HIERARCH ESO DET CLDC1 DC20  = 0.000 / Setup value for bias voltage (volt)
HIERARCH ESO DET CLDC1 DCT20 = 0.102 / Telemetry value for bias voltage
(volt)
```

Extension Header:

```
XTENSION= 'IMAGE   ' / Image extension
BITPIX  =                  -32 / # of bits per pix value
NAXIS   =                    2 / # of axes in data array
NAXIS1  =                  512 / # of pixels in axis1
NAXIS2  =                  512 / # of pixels in axis2
PCOUNT  =                    0 / PCOUNT
GCOUNT  =                    1 / PCOUNT
CRPIX1  =                  0.5 / Ref pixel in X
CRPIX2  =                  0.5 / Ref pixel in Y
EXTNAME = 'CHIP1.INT1' / FITS extension name
INHERIT =                    T / denotes the INHERIT keyword convention
HIERARCH ESO DET CHIP NAME   = 'myChipName' / Detector chip name
HIERARCH ESO DET CHIP ID     = 'myChipId' / Detector chip identification
HIERARCH ESO DET CHIP TYPE   = 'myChipType' / The Type of detector chip
HIERARCH ESO DET CHIP DATE   = '2005-08-03' / Date of installation
HIERARCH ESO DET CHIP PXSPACE= 1.000e-06 / Pixel-Pixel Spacing
HIERARCH ESO DET CHIP GAIN   = 1.00 / Gain in e-/ADU
HIERARCH ESO DET CHIP PSZX   = 1 / Size of pixel in X (mu)
HIERARCH ESO DET CHIP PSZY   = 1 / Size of pixel in Y (mu)
HIERARCH ESO DET CHIP XGAP   = 0 / Gap between chips along x (mu)
HIERARCH ESO DET CHIP YGAP   = 0 / Gap between chips along y (mu)
HIERARCH ESO DET CHIP RGAP   = 0 / Angle of gap between chips
HIERARCH ESO DET CHIP INDEX  = 1 / Chip index
HIERARCH ESO DET CHIP LIVE   = T / Detector alive
HIERARCH ESO DET CHIP X      = 1 / X location in array
HIERARCH ESO DET CHIP Y      = 1 / Y location in array
HIERARCH ESO DET CHIP NX     = 512 / number of pixels along x
HIERARCH ESO DET CHIP NY     = 512 / number of pixels along y
HIERARCH ESO DET FRAM STRX   = 1 / Start-X (lower left)
HIERARCH ESO DET FRAM STRY   = 1 / Start-Y (lower left)
HIERARCH ESO DET FRAM TYPE   = 'INT' / Frame type
HIERARCH ESO DET FRAM NO     = 1 / Frame number
HIERARCH ESO DET FRAM UTC    = '2006-09-13T08:14:10.4644' / File Creation
Time
HIERARCH ESO DET EXP UTC     = '2006-09-13T08:14:10.4309' / Time Recv Frame
```

Changes with respect to IRACE:

The mechanism of header and file creation is the same. FITS image extensions are supported by IRACE only to a limited extend (mosaics). The following FITS header keywords have changed their names:

DET.DIT	->	*DET.SEQi.DIT*
DET.MINDIT	->	*DET.SEQi.MINDIT*
DET.NCORRS	->	*DET.READ.CURID*
DET.NCORRS.NAME	->	*DET.READ.CURNAME*
DET.IRACE.SEQCONT	->	*DET.SEQi.CONT*
DET.RSPEED	->	*DET.SEQi.TIMEFAC*
DET.RSPEEDADD	->	*DET.SEQi.TIMEADD*
DET.EXP.NO	->	*DET.EXP.ID*
DET.EXP.UTC	->	*DET.FRAM.UTC*
DET.FRAM.UTC	->	*DET.EXP.UTC*
DET.CHIP.NO	->	*DET.CHIP.INDEX*
DET.VOLTi.xxxx	->	*DET.CLDCi.xxxx*
DET.WIN.STARTX/Y	->	*DET.FRAM.STRX/Y*
DET.FILE.CUBE.ST	->	*removed*
DET.WIN.TYPE	->	*removed*

The ADC category is not yet defined for NGC but will not be backwards compatible due to the new electronics design. The keywords not described in the above header example and not given in the changes-table are still TBD.

9.8. 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, see Figure 4) 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 glued image only for the real-time display. The ***DET.ACQ.SPLITX/Y*** keywords define how the chips are mapped in the glued image in x- and y-direction.

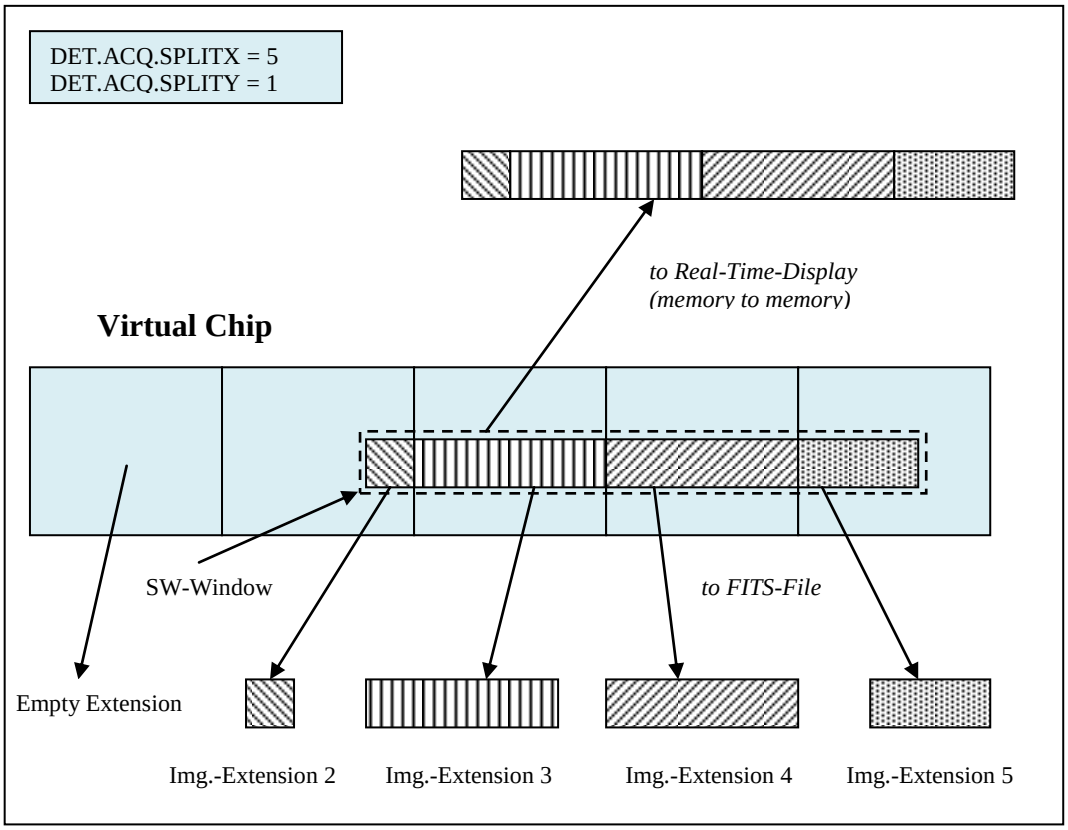

Figure 4 Virtual Chip and Overlapping Windows

Otherwise the ***DET.CHIPS*** keyword in the detector configuration file 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 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.

Changes with respect to IRACE:

The mosaic handling as done with IRACE is still possible (store to single FILES and merge by higher level SW). The file format "single-ext" must be used in this case to instruct the control server to add the additional FITS header keywords needed for the image extension format. In future versions this format may become obsolete (TBD).

9.9. Read-Out Windows

The dimension and position of the data-frame within the chip is defined by the setup parameters ***DET.SEQi.WIN.STRX***, ***DET.SEQi.WIN.STRY***, ***DET.SEQi.WIN.NX***, ***and DET.SEQi.WIN.NY***. ***DET.SEQi.WIN.STRX/STRY*** always refers to the lower left corner of the detector. For detector mosaics (***DET.CHIPS*** > 1, see section 9.8) 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 derived from the ***DET.SEQi.WIN*** parameters of the sequencer to which it is associated. When the read-out window changes then 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 defined independently. The parameters ***DET.ACQi.WIN.STRX/STRY***, ***DET.ACQi.WIN.NX/NY*** 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.

Software-windows (i.e. windows which are applied after the acquisition has been done but before the data is transferred) can be set individually for each frame type as described in sections 9.5 and 9.6. The ***DET.WIN.STRX/STRY***, ***DET.WIN.NX/NY*** parameters define a default software window to be applied to all frames. Furthermore these parameters are used to configure a window for the queue-mode (see section10.1.3). A zero value for the software window dimension generally indicates a "full-frame" read-out. The software window start coordinates always refer to the original detector coordinate system. The software window must fit into the read-out window.

Chip overlapping windows for virtual chips are taken into account as shown in Figure 4.

Changes with respect to IRACE:

The read-out windows are now applied "per sequencer". IRACE did not support multiple sequencers within a single system. The acquisition process window and the read-out window are now decoupled. The setup keywords have been changed accordingly to support the new features.

9.10. Exposure Driver

The NGC-DCS provides an exposure driving mechanism to run arbitrary exposure sequences within control server context. Each exposure is preceded by a configurable system setup. The driver can be selected by its public name and is configured through a configuration call-back function. The system provides a feed-back channel through which the driver can inform the outer world about the current actions taken. The driver has *read* access to all status keywords and *read/write* access to the data frames received during the exposures. The processing can be done directly on the received data-frames and the processed data can be stored. For advanced operations the driver has direct access to the controller functions (use with care).

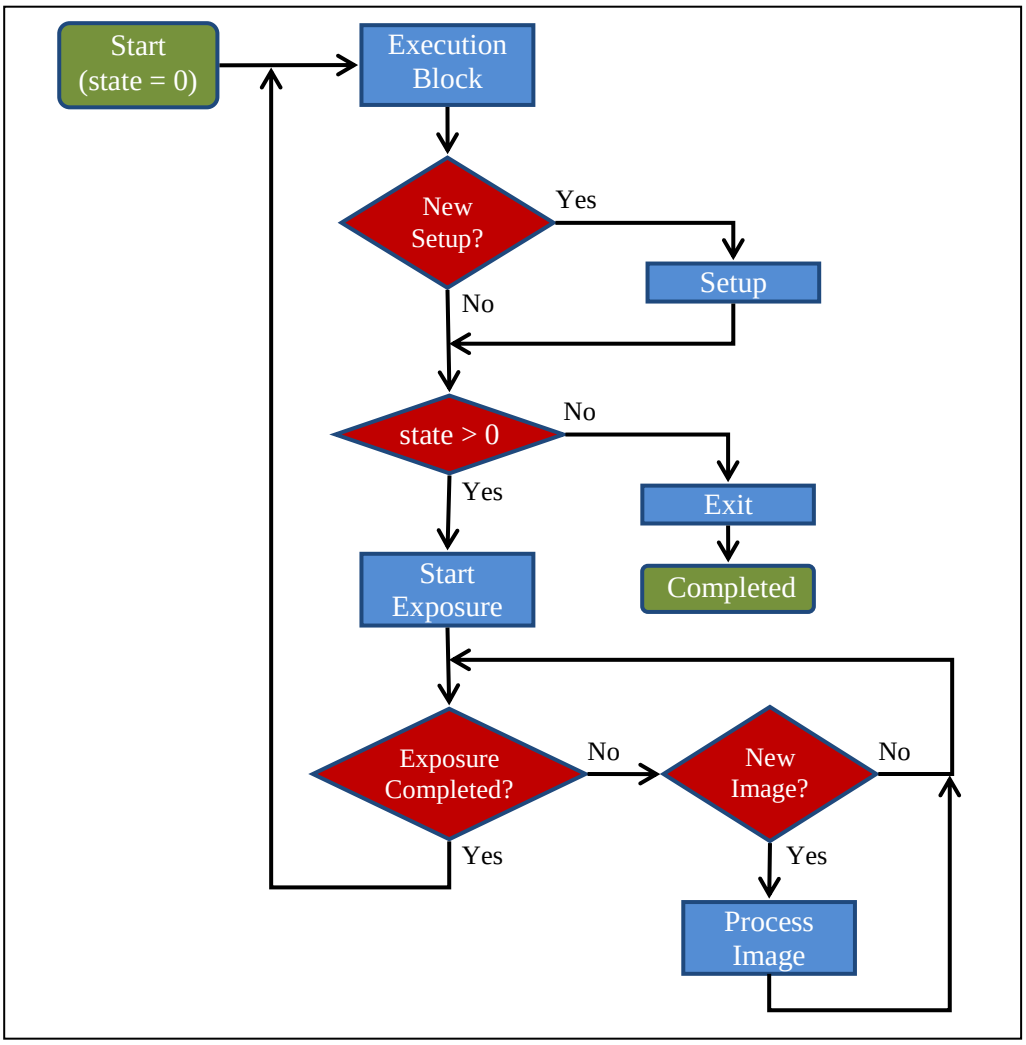

Figure 5 Exposure Driver Control Flow

9.10.1. Database

As indicated in section 6.6 the actual exposure driver status is available in the database attribute '***<alias>ngcircon:exposure.drvStatus***'. It can take one of the following values:

```
0   -  none (no driver installed)
1   -  inactive (idle)
32  -  processing (active)
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.drvStatusName***'.

9.10.2. Commands

The exposure driver can be controlled with the ***EXPDRV*** command (see also section 4). It is possible to load external exposure driver modules with the "*-load <module>*" option. The drivers must be packed in a shared library with the given string as library name ("*lib<module>.so*"). The library must be installed in the shared library search path (*$LD_LIBRARY_PATH*). Driver modules loaded with the "*-load*" option can be unloaded again with the "*-unload <module>*" option. Driver modules can be unloaded only when none of the exported drivers is in use.

Both built-in and dynamically loaded drivers can be installed with the "*-select <name>*" option and can be uninstalled with the "*-delete*" option. The "*-select*" option automatically uninstalls a previously selected instance. The name of a dynamically loaded driver is not necessarily identical with the module name used for the "*-load*" option. The "*-list*" option either returns a list of the dynamically loaded modules ("*-list mod*") or it returns a list containing the names of all selectable drivers ("*-list drv*") including the ones exported from dynamically loaded driver modules. The names are separated by <*newline*>.

The "*-config ...*" option configures the driver by pairs of *parameter-name* and *parameter-value*. When no parameters follow the "*-config*" option then the command returns the current configuration in the reply. When only one word follows the "*-config*" option then this word is treated as the name of a file from which the configuration parameters are loaded. The file format is *SHORT-FITS*. The scope of the file is the *host* and the *environment* of the control server.

The exposure sequence is started with the "*-start*" option and can be aborted with the "*-abort*" option. The exposure sequence only continues when the running exposure completes successfully. This means that the global ***ABORT*** command (section 9.2) also terminates the sequence while the global ***END*** command (section 9.2) just shortens the sequence by jumping directly to the completed state.

A feed-back channel can be selected by name with the "*-fb <name>*" option. Supported values are *"-fb none*", "*-fb mon*", "*-fb stdout*", "*-fb stderr*" and "*-fb log*". The "*-fb mon*" option opens a new terminal where the feed-back messages are displayed. The "*-fb log*" option forwards the feed-back messages to the logging system.

The feed-back channel and the driver can also be selected or queried with the ***SETUP/STATUS*** keywords ***DET.EXP.DRVFB*** and ***DET.EXP.DRVNAME*** (see section 7.1).

The ***STATUS*** keyword ***DETDET.EXP.DRVSTAT*** can be used to retrieve the current state of the exposure driver (see also section 8.1). The state can be one of "*inactive*", "*processing*", "*success*", "*failure*" or "*aborted*". An empty name or an empty state-string both indicate that no driver is currently installed. The keyword ***DET.EXP.DRVMSG*** stores a message issued by the driver when its state changes to "*failure*" or "*aborted*". The message is cleared whenever a new exposure sequence is started,

9.10.3. Driver Tool

```
ngcdcsExpDrv [name] [-host <host>] [-port <N>] [-gui] [-v [level]] ]
                    [-log [level]] [-cfg …]

ngcdcsExpDrv [name] –server <server-name> [-gui] [-v [level]]
                    [-log [level]] [-cfg …]
```

In its first variant the exposure driver tool establishes a connection to the NGC-DCS server on the given host/port. If the *-host* option is not specified then the environment variable ***$HOST*** is taken for the host and ***$NGC_PORT*** is taken for the port number of the control server.

In the second variant the server communication is done through the CCS message system and the (mandatory) *-server* option defines the destination process name (e.g. "*-server ngcdcsEvh*"). It is possible to specify the destination environment when the *server-name* has the format "*env:proc*" (e.g. "*-server wdcs:ngcdcsEvh*"). The default destination environment is defined by the environment variable ***$RTAPENV***.

When the driver *name* is present in the command line it must be the first argument. The tool then installs the exposure driver for *name* and all parameters following the *-cfg* option will be forwarded to the exposure driver configuration list. The configuration options are given as pairs of *parameter name* and *parameter value*.

Example: "***-cfg param1 value1 param2 value2 … paramN valueN***"

When the *-cfg* option is followed by only one word then this word is treated as the name of a file from which the configuration parameters are loaded. The file format is *SHORT-FITS*. The scope of the file is the *host* and the *environment* where the *ngcdcsExpDrv* tool has been launched.

When the *-cfg* option is given without any further parameter then the tool just prints the current configuration to *stdout* and exists.

When the *-gui* option is present then the exposure driver GUI (see section 9.10.4) is launched. Otherwise the exposure sequence is started immediately and the feed-back messages indicating the exposure sequence progress are written to *stdout*. When no name is given as first argument in the command line then the GUI is always launched in order to select a valid driver.

When the tool is interrupted (e.g. by a signal) then it will send an ***ABORT*** command to the NGC-DCS to abort the running exposure sequence.

After completion the tool uninstalls the exposure driver again.

When the *-gui* option is not present then the internal verbose messages of the exposure driver tool are by default enabled (verbose level 1). The verbose messages can be disabled by setting the verbose level to zero with the *-v* option.

The *-log* option enables the creation of log-file messages if the *level* is greater than zero.

9.10.4. Driver GUI

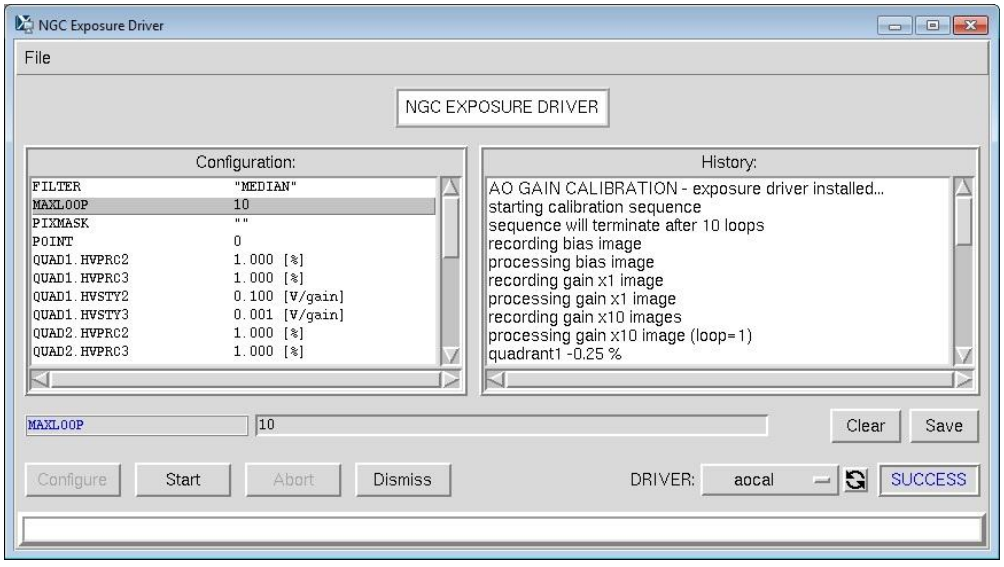

Figure 6 Exposure Driver GUI

10. Data Buffering

10.1. Burst Modes

In some cases it is 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. Three kinds of "*burst modes*" are provided to cover these cases.

10.1.1. 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. The transfer of the sample-frames starts immediately and runs in parallel to the data recording. 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 acquisition process as defined for the current read-out mode.

The data type of the produced data is 16-bit unsigned short integer (this is the format delivered by the NGC analog-digital converters). This is changed to BITPIX=16 (signed integer) in the FITS data file with BZERO set to 327678. Signed integer is the only 16-bit integer format supported by the FITS standard.

The default burst process (*ngcppBurst*) applying the described mechanism may be changed by specifying a user-defined process to be used in burst-mode (***DET.ACQi.BURST.PROC***). This may imply a change in the data type also.

10.1.2. 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. The DMA is enlarged by a factor of ***num*** and soft-interrupts are created for each sub-division. The data processing is not affected. This helps to work around the frame rate limitation but depending on the actual processing algorithms the overall performance may be slowed down. Also in this case a value of zero for ***DET.ACQi.BURST.NUM*** deactivates the burst-mode.

10.1.3. Queue Mode

This mode implements a large frame queue similar to the "*raw-data mode*" described in section 10.1.1. The difference is that the frames are queued after the normal pre-processing and pixel sorting. The queue mode can be used when the CPU-bandwidth of the NGC-LLCU is large enough to do the pre-processing but the network is too slow to transfer all generated frames in time to the instrument workstation. The mode is activated by sending the ***SETUP*** command

"*SETUP -function DET.ACQi.QUEUE <num>*"

The value of ***num*** indicates the number frames to be queued. A zero value deactivates the queue mode. Only one frame type can be put to the queue which is always the type with the highest frequency. This may be the ***SAMPLE***-, ***DIT***- or ***INT***-frame depending on the selected acquisition process. Only this frame type can be stored during the exposure. All other frames are not queued but are always available for display. When none of these three types is produced by the acquisition process the queue mode has no further effect.

Examples:

- Acquisition process produces ***DIT***- and ***INT***-frames:

  "*Then **DET.ACQi.QUEUE** subsequent **DIT**-frames will be stored during the exposure. All other frames are available only for display.*"

- Acquisition process produces ***SAMPLE***-, ***DIT***- and ***INT***-frames:

  "*Then **DET.ACQi.QUEUE** subsequent **SAMPLE**-frames will be stored during the exposure. All other frames are available only for display.*"

A window can be defined for the queued frame via the ***DET.WIN.STRX/STRY/NX/NY*** parameters (see section 9.9). The generation of the "*queue-frame*" automatically stops after ***DET.ACQi.QUEUE*** images have been processed and is resumed at the start of the next exposure. So once the queue is filled the "*queue-frame*" will not be displayed until the next exposure is started (see section 10.4.2 for an exception when using the "*queue dump*" instead).

The queue mode can still be combined with the "*internal burst mode*" (section 10.1.2). In "*raw-data mode*" (section 10.1.1) the value of ***DET.ACQi.QUEUE*** has no meaning.

Changes with respect to IRACE:

The burst mode can now set individually per acquisition module:

DET.BURST.NUM -> DET.ACQi.BURST.NUM
DET.BURST.SKIP -> DET.ACQi.BURST.SKIP
DET.BURST.PROC -> DET.ACQi.BURST.PROC

10.2. Memory Extension

In both burst- and queue-mode data can be buffered in memory up to a size of 3 Gbytes (32-bit Linux limitation). To overcome this limit the NGC-LLCU local disk can be used to extend the buffering capacity. A ***raid0*** configuration with at least 4 parallel disks and a non-journaling file system (e.g. "***ext2***") are recommended to achieve a good performance (measured bandwidth in this configuration: up to 240 Mbytes/s sustained data transfer). The disk-space must be pre-allocated once:

```
SETUP -function DET.ACQi.DMSIZE <size>
```

(the *size* is given in Mbytes)

The storage directory can be defined via the setup command:

```
SETUP -function DET.ACQi.DMPATH <path>
```

When the keyword is not defined then the default path "*/tmp*" is used. All VLT-Linux installations support the usage of a RAM disk on the location "*/dev/shm*". The size is half of the physically installed memory (i.e. typically the RAM disk size is 8 Gbytes). When not exceeding the RAM disk capacity this is the fastest way of extending the buffering memory.

The disk memory extension can be enabled or disabled with:

```
SETUP -function DET.ACQi.DISKMEM T/F
```

The disk-space remains allocated until it is explicitly freed by setting the *size* to zero. The disk-space allocation survives a control server restart. Disk access bottle-necks can be overcome by increasing the number of frames stored in the DMA ring-buffer (use acquisition-process option "*-nbuf <N>*"). This is a valid option here because most of the process space (non-DMA) memory is swapped out to disk.

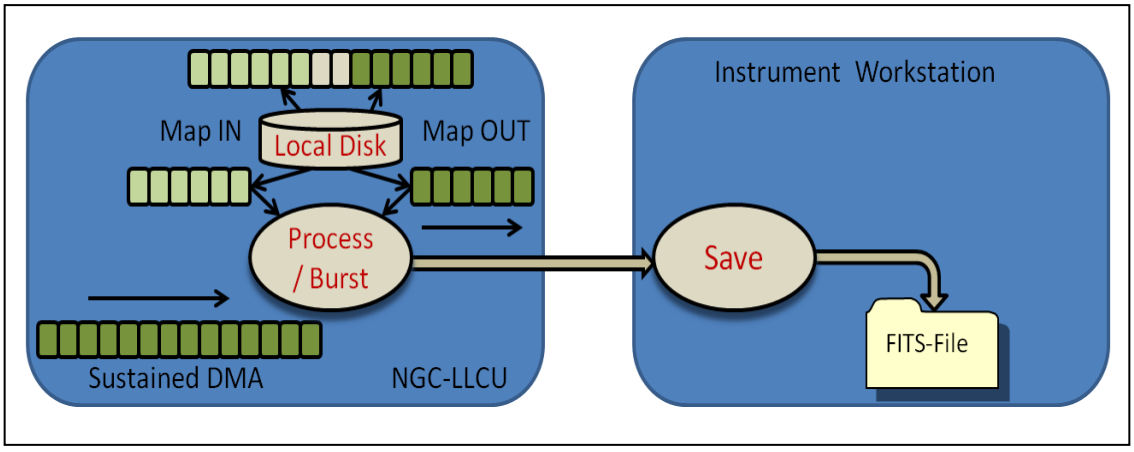

Figure 7 Disk Memory Extension

10.3. Large Files

In *Queue-Mode* and *Burst-Mode* frames are typically stored in data-cubes (NAXIS3 = number of frames). The normal file size limit is 2 Gbytes (`fopen()`, ANSI standard). When the limit is exceeded a data-cube can be split into several files.

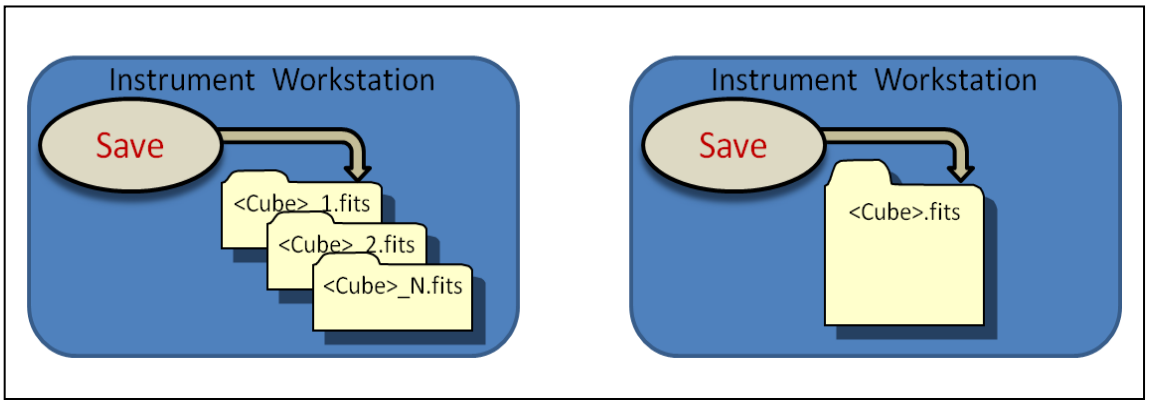

Figure 8 Large Files

Caution:
Archive identifiers are constructed as "instrument.MJD-OBS". Archiving multiple data cube files (which all have the same MJD-OBS) requires some workaround.

The 64-bit "*Large-File-Extension*" for 32-bit Linux allows large files but requires special functions (`fopen64()`, LFS – "*Large File Support*") and/or compilation flags:

- Not many applications are able to deal with 64-bit file offsets.
- No possibility to keep the whole dataset in memory.
- Parallelization (post-processing) can only start when the whole file is saved.

For DCS the large-file extension is configurable via the keyword:

```
setup -function DET.FRAM.LRGFILE T/F
```

Typically this keyword is defined in the system configuration file.

10.4. Overheads

The burst- and queue-modes imply a significant transfer overhead. Even though the transfer starts in parallel after having produced the first chunk of data it will still take a long time until the whole buffer is flushed to the disk on the instrument workstation. Once all data is buffered the exposure immediately goes to "***transferring***"-state (see section 9.1) thus allowing to change the field of view. But still the transfer overhead limits the rate of consecutive exposures as the next exposure can only be started once the previous one has finished.

Caution:
This section contains preliminary/incomplete information. The contents shall not be used for interface control documentation. The compliance of the reference and merging mechanisms with other control SW specifications still has to be proven.

10.4.1. Local Storage

To overcome the transfer overhead the pre-processor provides the possibility to store the data permanently on the local storage media of the NGC-LLCU (local disk, RAM disk, etc.):

```
setup -function DET.FRAMi.LOCAL T
```

The ***FRAMi*** index here refers to the respective frame type (e.g. ***SAMPLE*** or ***DIT***, see section 9.6).

Then on the instrument workstation only FITS-header files (with the extension "*.hdr*") are generated which contain a reference to the physical location of the data.

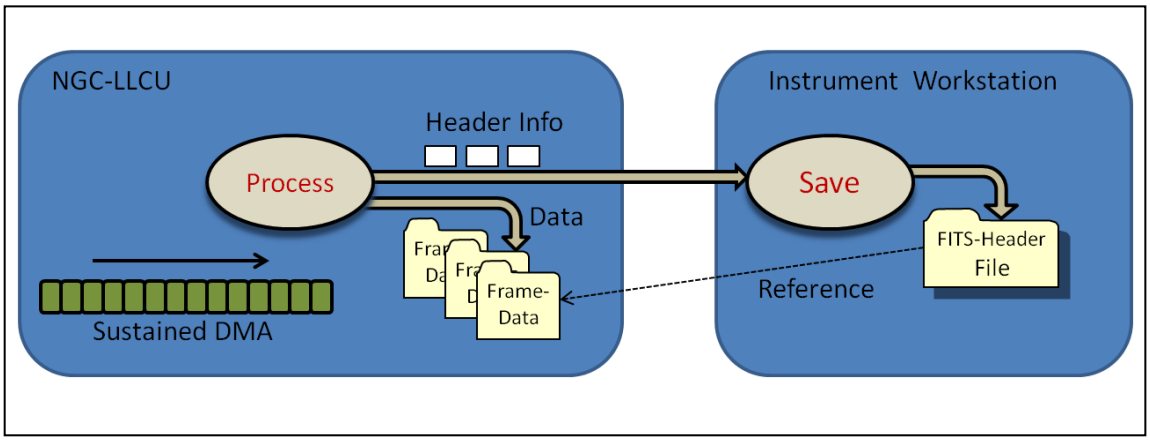

Figure 9 Local Storage

The physical location of the local storage area can be defined via the keyword:

```
SETUP -function DET.ACQi.LPATH <path>
```

The default path is defined by the ***$NGCPP_DPATH*** environment variable on the NGC-LLCU. When the environment variable is not defined then "*/tmp*" is used by default. The path can be reset to its default value by setting the ***DET.ACQi.LPATH*** keyword to the string "*default*". The data is still buffered in memory before being written either to local disk or to the network link depending on the setting of the ***DET.FRAMi.LOCAL*** keyword. So this approach is still not sufficient when the memory limitation is reached.

10.4.2. Queue Dump

In order to buffer directly to disk a so-called "*dump-mode*" has to be activated:

```
setup -function DET.ACQi.QDUMP T/F
```

This will instruct the pre-processor to generate permanent data files for all buffers in the queue (section 10.1.3) or in the burst (section 10.1.1). The physical location is again set with the ***DET.ACQi.LPATH*** keyword. Once all data-buffers are written to the local storage media a notification event is sent to the control server which then terminates the exposure and again generates a FITS header file (extension "*.hdr*") with a reference to the data. As the buffer files are never overwritten the associated queue frames buffered in memory are always available for the display.

The "*queue dump*" is only valid in queue-mode (section 10.1.3) or in raw-data burst-mode (section 10.1.1). In all other cases it has no further effect. As with the disk memory extension (see section 10.2) the disk access bottle-necks can be overcome by increasing the number of frames stored in the DMA ring-buffer (use acquisition-process option "*-nbuf <N>*"). Also here this is a valid option because most of the process space (non-DMA) memory is swapped out to disk.

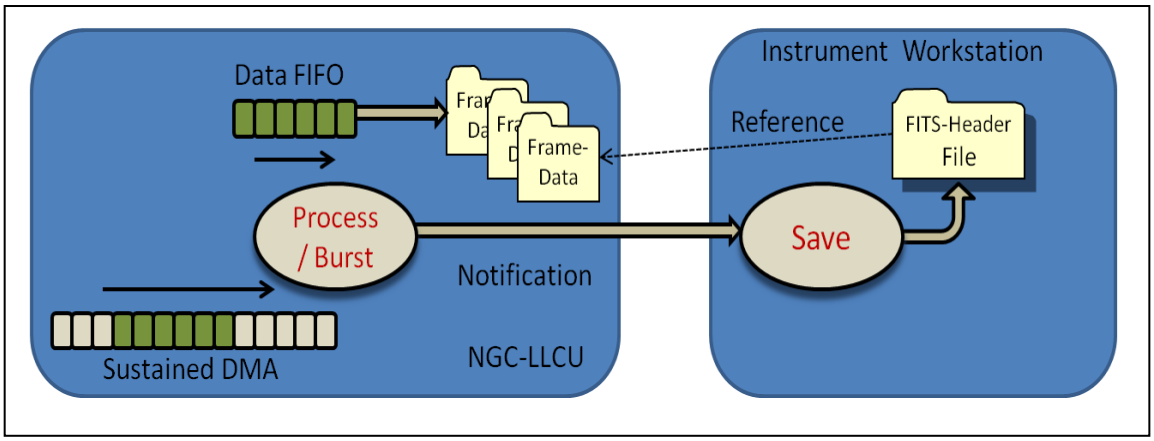

Figure 10 Queue Dump

10.5. Data Merger

Caution:
This section contains preliminary/incomplete information. The contents shall not be used for interface control documentation. The compliance of the reference and merging mechanisms with other control SW specifications still has to be proven.

The methods proposed in section 10.3 require an additional asynchronous task to finally copy the data residing on the NGC-LLCU local storage media to their proper place in the FITS-file on the instrument workstation once the exposure is completed. The ***ngcdcs*** module provides a merging tool which reads the file reference from the FITS-header and then merges the header with the frame data.

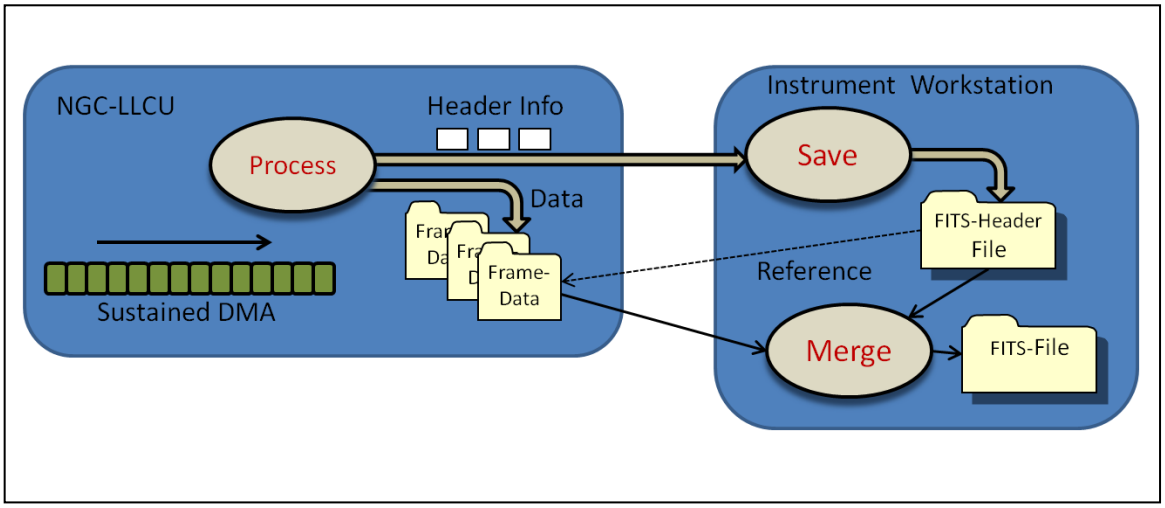

Figure 11 Data Merging

The merger can be launched:

a) At any time by explicitly sending a command (***MERGE***) to the control server (see section 10.5.1). The calling application needs to keep track of the merging jobs still to be done.

b) As an automatic procedure after exposure completion (see section 10.5.5). The control server keeps track of the merging jobs still to be done.

As this is typically used with rapid sequences of exposures the merging may lag behind a considerable amount of time and the list of jobs can become large (up to several thousand files). *TBD: How to make this list public? How to recover the list after a process crash?*

10.5.1. Commands

The ***MERGE*** command can be sent to the control server to initiate or abort the merging tasks. The system supports up to four merging tasks running in parallel.

MERGE	**-name**	String	Exposure base name or FITS-file name to be completed. When no explicit id is requested then a new merger id will be returned.
	-id	Integer	Explicitly request a merger id (1-4). When not specified the next available merger task will be used.
	-abort	Logical	Issue abort request to stop the merger with the given id. This requires an explicit id to be specified.
	-cleanup	Logical	Cleanup temporary files.
	-display	String	Indicates whether or not the merger process should launch a status display. The default value for this option is "T". When the option is not present then the value of the DET.MERGE.DISPLAY parameter is used.
	-resume	Logical	Resume an interrupted auto-merging procedure.
	-clear	Logical	Clear auto-merger list.

Table 28 MERGE command

The ***MERGE*** command returns the *merger-id* in use. The "*-resume*" function always returns zero. The command can also be sent from within the engineering GUI (section 15.1):

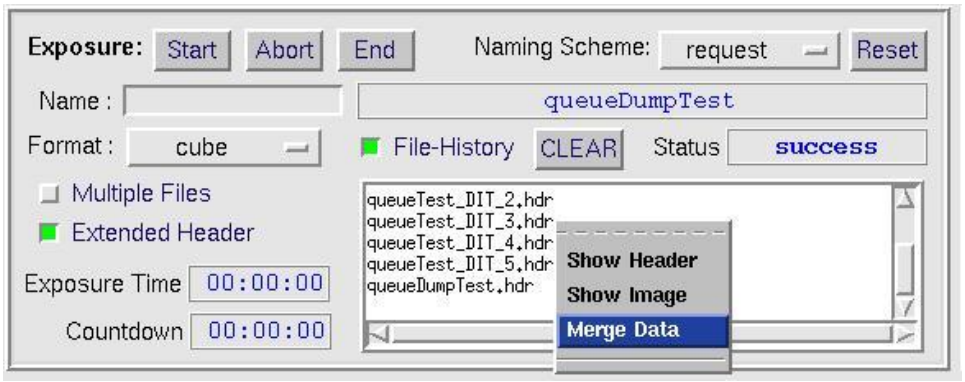

Figure 12 Start Merging from GUI

10.5.2. Parameters

The following ***SETUP*** and ***STATUS*** parameters are available to tune the merging:

Keyword	Type	Description
DET.MERGE.DISPLAY	Logical	Launch status display when merging.
DET.MERGE.INVLDT	Logical	Invalidate source files after merging. Default value is "F".
DET.MERGE.VERIFY	Integer	Verify file consistency before merging. Default is "T".
DET.MERGE.CHUNK	Integer	Define a chunk size for merging. When set to zero then the chunk size is always set to the full frame. A negative value indicates that the file-merger default value should be used.
DET.MERGE.AUTO	Logical	Enable/disable auto-merging (see section 10.5.5).
DET.MERGE.CONCUR	Logical	Enable/disable concurrent merging. When disabled then only one merging process can run at a time.

Table 29 Merger Setup and Status Keywords

Additionally the actual status (string value) of each merging task can be retrieved via the ***DET.MERGEi.STATUS*** keyword. The index refers to the *merger-id* as returned by the ***MERGE*** command (Table 28).

The merger status can be one of:

Name	Integer Value
"***idle***"	0
"***pending***"	1
"***active***"	2
"***verifying***"	4
"***merging***"	8
"***invalidating***"	16
"***success***"	128
"***failure***"	256
"***aborted***"	512

Table 30 Merger Status

10.5.3. Database

One instance ("*merger_<i>*") of the ***ngcdcsMERGE*** database class (see Table 48) is used per merger task. The index refers to the *merger-id* as returned by the ***MERGE*** command (Table 28) minus one (database indexes always start with zero).

Example (*merger-id* = 1): *<alias>ngcircon:merger_0.<attribute>*

Attribute	Type	Description
pid	INT32	Merger process id.
status	INT32	Merger status.
statusName	BYTES32	Merger status converted to a string.
abort	INT32	Abort request flag. When the flag is raised by an application then the merger will abort.
vsize	DOUBLE	Expected size after verification run (in bytes).
tsize	DOUBLE	Transferred size (in bytes).
newDataFileName	BYTES256	New data file.
msg	BYTES256	Status/error message.

Table 31 Merger Database Class (ngcdcsMERGE.class)

Additionally each merger instance will update the global exposure new-data-file attribute '***<alias>ngcircon:exposure.newDataFileName***' whenever a new file is completed. *TBD: Do this only when auto-merging is enabled?*

10.5.4. Merger Status Display

Optionally the merger will launch an internal status display:

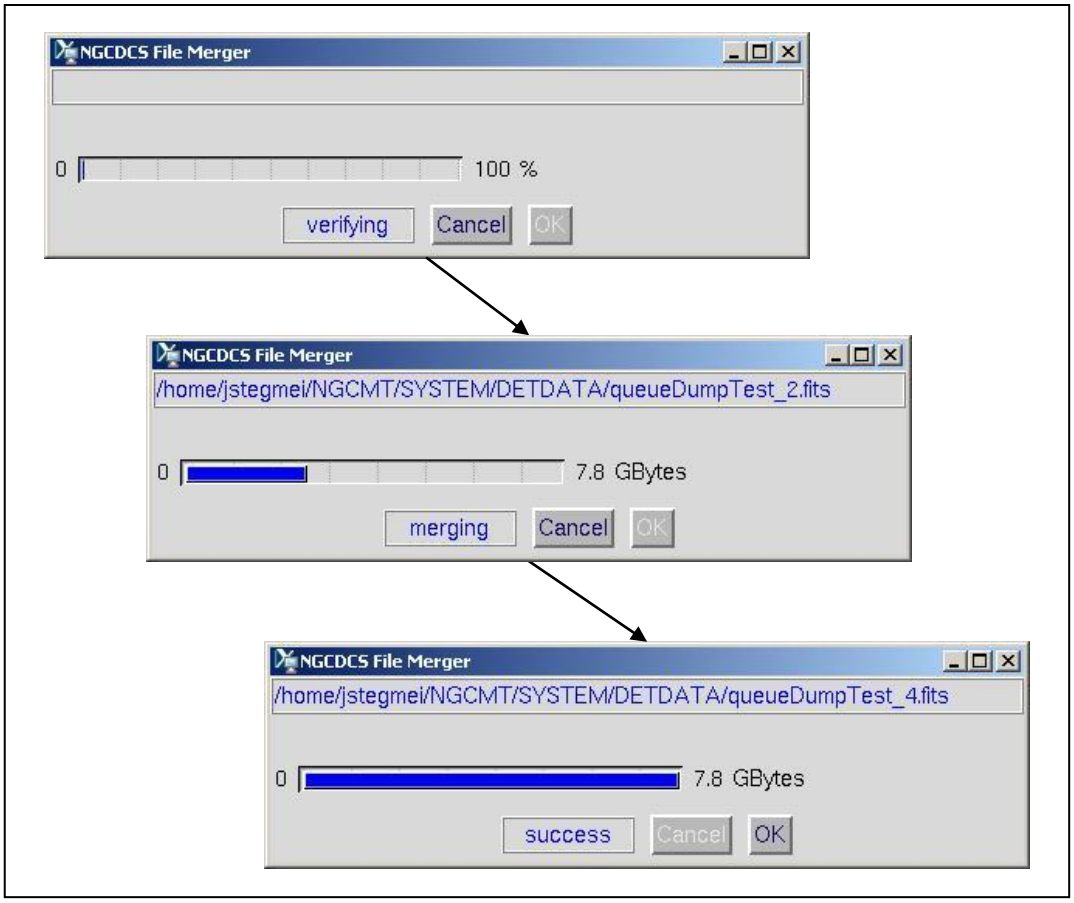

Figure 13 Merger Internal Display

10.5.5. Automatic Data Merging

The merging can be done automatically by the control server each time after an exposure is completed. When all mergers are busy at this time then the merging-jobs are put to a queue-list (FIFO). Each time when one of the mergers has completed its current task then the next job in the queue will be processed. This is done until the list becomes empty. When the list is empty the merging will be resumed with the next successfully completed exposure.

The control flow of the automatic merging is shown in Figure 14.

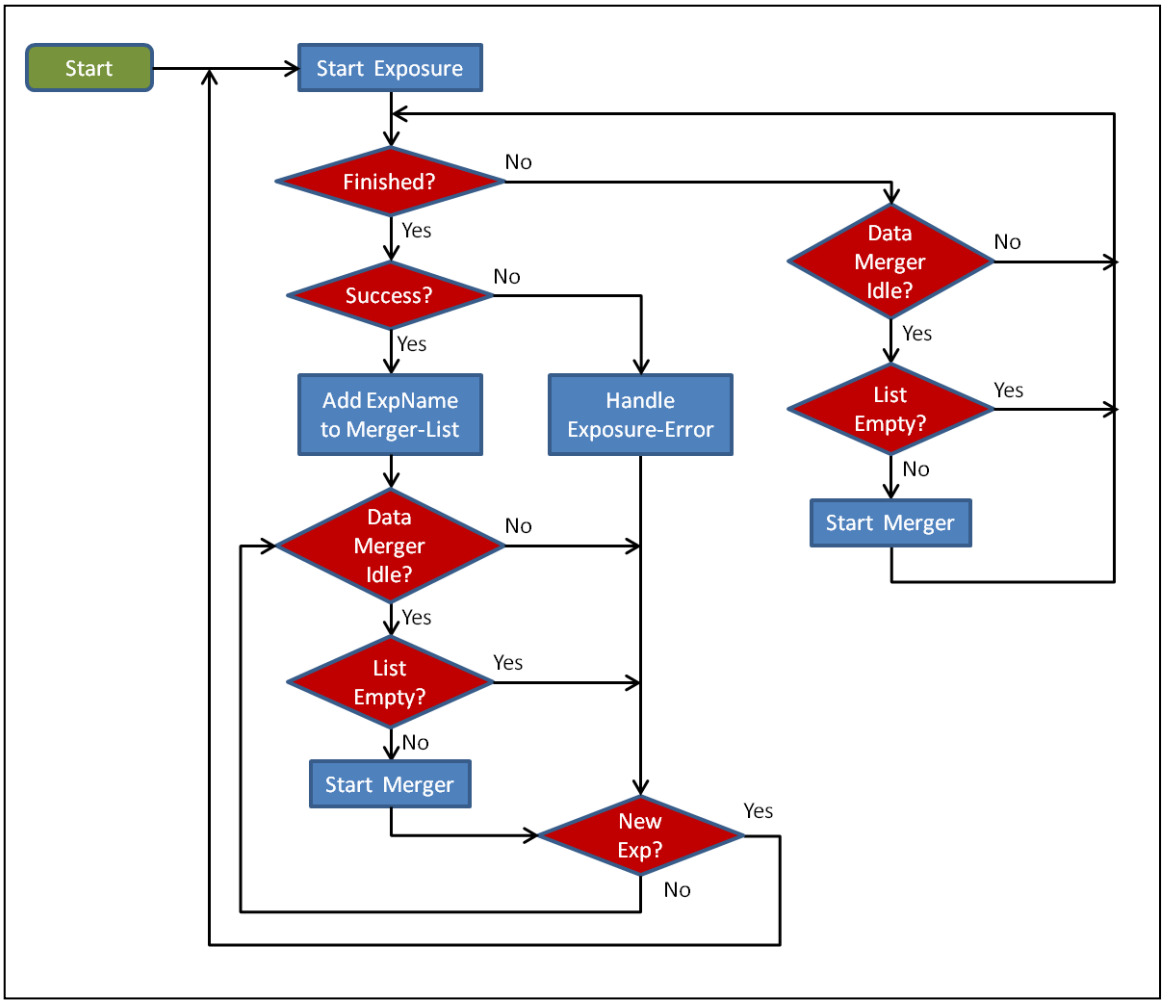

Figure 14 Auto-Merger Control Flow

The automatic merging is controlled via the ***DET.MERGE.AUTO*** parameter:

```
SETUP -function DET.MERGE.AUTO T/F
```

When the automatic merging is enabled then the global exposure new-data-file attribute '***<alias>ngcircon:exposure.newDataFileName***' will not be updated when the exposure completes. The file is given to the merging procedure instead which will

then in turn update the database upon successful file completion. The intermediate file event ("*.hdr*" file) after exposure completion can still be traced by attaching to the additional '***<alias>ngcircon:exposure.fileEvent***' attribute.

When the merging fails the corresponding error message is saved to the job entry in the merging queue list. When the merging is aborted the job-entry is marked as aborted (*TBD: retry?*). When the merging completes successfully then the job entry is removed from the list. *TBD: How to make the list entries public? How can we resume merging when the server is restarted?*

The merger internal status display (section 10.5.4) is not practical within an automatic procedure. A static panel is available to see the current status of all mergers in one view:

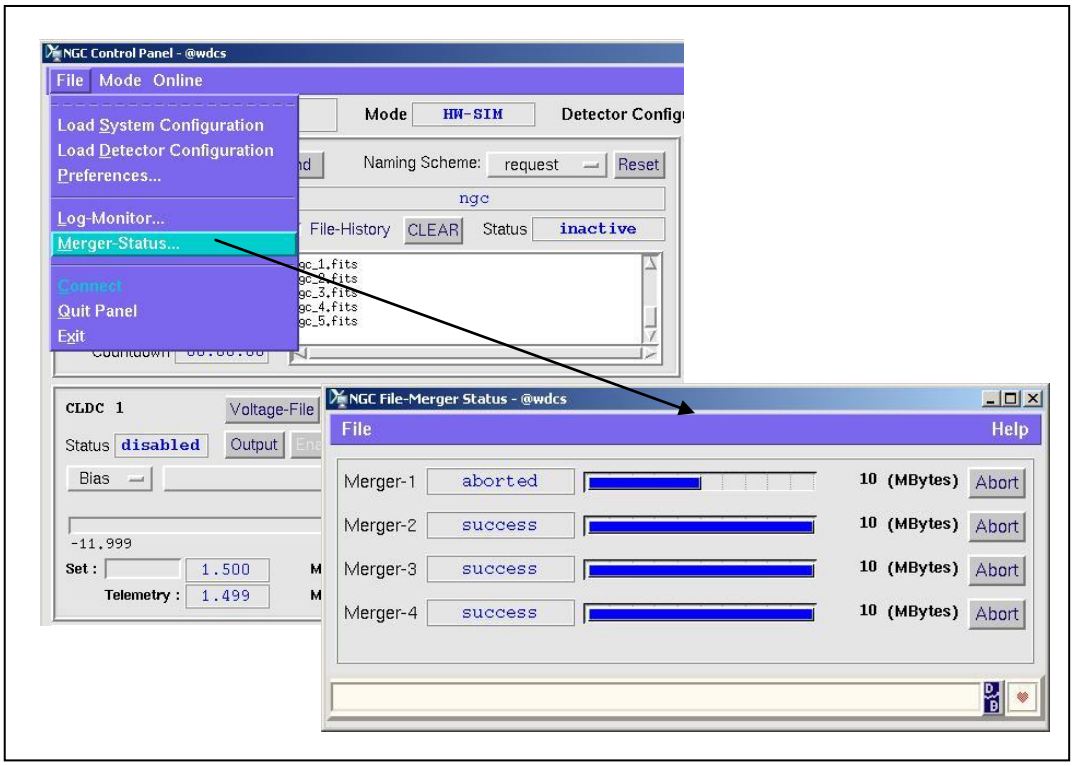

Figure 15 Merger Status Display

11. Synchronization

Synchronization points can be inserted at any place in any clock pattern executed by the sequencer program by setting the "*wait-for-trigger*" bit (***DET.PATi.CLK61***). See section 16.1 for clock pattern definition and section 16.2 for sequencer programming.

When reaching such a state where ***DET.PATi.CLK61*** is set to '1' the pattern execution is suspended until the arrival of an external trigger signal (see [AD6] and [RD1] for signal timing and accuracy).

Example (four-state-pattern with synchronization at the end):

```
DET.PAT4.NAME   "FrameSync";
DET.PAT4.NSTAT 4;
DET.PAT4.CLK1   "1100";
DET.PAT4.CLK2   "1010";
DET.PAT4.CLK3   "0110";
      :
DET.PAT4.CLK61 "0001";       # SYNC
DET.PAT4.DTV    "5,5,5,5";    # Dwell-Time vector
DET.PAT4.DTM    "1,1,1,1";    # Dwell-Time modification flags
```

The "***waiting***" state is only activated when the keyword ***DET.SEQi.TRIGGER*** is set to "***T***". Otherwise the "***waiting***" state is just passed through.

Via the external trigger input it is possible to synchronize exposures on multiple *NGCIRSW* instances. The external trigger input is also used to synchronize detector read-outs with external devices (e.g. chopper, see section 22). Using the VLT-TIM for generating the trigger pulse(s) allows synchronization at absolute times.

Furthermore some of the signal lines are available to in turn trigger external devices (e.g. tell another device that a read-out has finished).

The trigger signal is not queued. A trigger which is sent when the sequencer is not in "***waiting***" state will be lost.

Caution:
The sequencer will go to "idle" state when the pattern address FIFO gets empty. So if the "wait-for-trigger" bit is set in the last pattern of a finite sequence (i.e. when no other pattern follows the "waiting"-pattern) the pattern-address FIFO becomes empty and the sequencer goes to "idle" state instead of the expected "waiting" state. Finite sequences using the trigger synchronization should always terminate with a pattern not having the "wait-for-trigger"-bit set.

The trigger can also be sent via command:

"*SEQ -trigger -module <n>*"

Here <***n***> is the sequencer instance number. The command will synchronously trigger all sequencers which have the external run-control enabled (***DET.SEQi.RUNCTRL = T***). If ***n = 0*** (default) all modules are triggered (but not necessarily synchronously).

If several sequencers are installed in the same system (i.e. the same instance of *NGCIRSW*) then the exposure start 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***). If one of the synchronous sequencers is not operating in continuous mode (i.e. it is stopped and restarted at exposure start – see section 9.2), then the ***DET.SEQi.CONT*** flag is also ignored for all other synchronous sequencers.

If no high accuracy is needed then the exposure start synchronization can be also done at command interface level (e.g. issue an exposure start command at the proper time or use the command "*START –at <start-time>*" as described in sections 4 and 9.2).

Changes with respect to IRACE:

The SW-handling of the external trigger pulse is backward compatible with IRACE. The actual signal shape + level and also the connector have changed. IRACE did no support multiple sequencers within a single system.

12. Error Definitions

The CCS error mechanism [RD5] provides a classification scheme for application specific errors. The introduction of new error codes is limited to cases, where specific actions (like "*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. Without doing system interaction outside the scope of *NGCIRSW* (e.g. controller electronics check, computer reboot, environment restart) the first recovery procedure is always to send an ***ONLINE*** command to the control server and leave it up to the SW do the proper steps. If this does not cure the problem and the reason is still assumed to be inside *NGCIRSW*, then generally a server restart is required.

Table 32 and Table 33 show the error definitions used by *NGCIRSW*:

Error	**Severity**	**Description**
`ngcbERR_SYSTEM`	Warning	General error of informative character ("*parameter out of range*", "*invalid file name*", etc.).
`ngcbERR_IO`	Serious	An I/O-error on the interface to the detector front-end occurred. Typically this will require at least a reset (*STANDBY – ONLINE*) or a power-cycle of the NGC controller electronics to recover.
`ngcbERR_SRV`	Serious	The communication with the driver interface process failed. The server may have died or the message system/network is down. Go *LOADED-STANDBY-ONLINE* to recover from this.
`ngcbERR_INIT`	Serious	The server initialization failed. The message system may not be up, or the operating system has run out of its resources. This may require a restart of the environment or even a reboot of the IWS/NGC-LLCU.

Error	Severity	Description
ngcbERR_WARNING	Warning	A warning which has only very limited effect on the further system behavior.
ngcbERR_DB_READ	Serious	Error when reading from the online database. This may require a rebuild of the database and a restart of the CCS environment.
ngcbERR_DB_WRITE	Serious	Error when writing to the online database. This may require a rebuild of the database and a restart of the CCS environment.
ngcbERR_DB_INIT	Serious	Error when accessing the online database during initialization phase. This may require a rebuild of the database and a restart of the CCS environment.

Table 32 Error Definitions (*ngcb* SW module)

Error	Severity	Description
ngcdcsERR_SYSTEM	Warning	General error of informative character ("*parameter out of range*", "*invalid file name*", etc.)
ngcdcsERR_IO	Serious	An I/O-error on the interface to the detector front-end occurred. Typically this will at least require a reset (*STANDBY – ONLINE*) or a power-cycle of the NGC controller electronics to recover.
ngcdcsERR_INIT	Serious	The server initialization failed. The message system may not be up, or the operating system has run out of its resources. This may require a restart of the environment or even a reboot of the IWS/NGC-LLCU.
ngcdcsERR_WARNING	Warning	A warning which has only very limited effect on the further system behavior.
ngcdcsERR_FATAL	Fatal	An error which cannot be recovered in any case.
ngcdcsERR_ACCESS	Serious	The function could not be executed due to missing access rights. Some commands are allowed for test and development purposes only.
ngcdcsERR_DB_READ	Serious	Error when reading from the online database. This may require a rebuild of the database and a restart of the CCS environment.
ngcdcsERR_DB_WRITE	Serious	Error when writing to the online database. This may require a rebuild of the database and a restart of the CCS environment.
ngcdcsERR_DB_INIT	Serious	Error when accessing the online database during initialization phase. This may require a rebuild of the database and a restart of the CCS environment.

Error	Severity	Description
ngcdcsERR_ACQ_IO	Serious	An I/O-error occurred when communicating with the acquisition process. The process may have died or the network connection to the NGC-LLCU may be broken. Go *STANDBY-ONLINE* to recover.
ngcdcsERR_ACQ_OVERRUN	Serious	The acquisition process was not able to process the data in time. All data are buffered in a ring-buffer to compensate operating system jitters. If data are coming in faster than they can be processed for a longer period of time, then the ring-buffer may overrun. Go ONLINE again and either read-out slower or add sufficient delays between the read-outs. The error can also be recovered by just stopping and restarting the acquisition ("*SEQ –stop*" / "*SEQ –start*" commands).
ngcdcsERR_ACQ_OVERFLOW	Serious	The data on the physical NGC-data link are coming in faster, than they can be delivered to the computer bus. The internal FIFOs on the interface boards become full in this case. This is a controller electronics error. The reason may be for example crosstalk/spikes on the physical communication lines. Check the controller electronics and go ONLINE again to recover. The error can also be recovered by just stopping and restarting the acquisition ("*SEQ –stop*" / "*SEQ –start*" commands).
ngcdcsERR_ACQ_EXEC	Serious	The acquisition process could not be executed on the target host. Process-name, search-path or permission may be wrong.
ngcdcsERR_EXP_IO	Serious	An error occurred during data taking. The data connection to the acquisition process may be broken (process has died or network is down). Try to go ONLINE again to recover.
ngcdcsERR_EXP_FILE	Serious	An error occurred when writing the exposure data to a file on the disk. The disk may be full or the directory can no more be accessed.
ngcdcsERR_EXP_PROC	Serious	An error occurred within the post-processing call-back.

Table 33 Error Definitions (*ngcdcs* SW module)

Changes with respect to IRACE:

The error definitions are not backwards compatible with IRACE.

13. Error and Logging Handling

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 [AD10], [RD5]). Additionally the verbose output can be logged in a detail depending on the given log-level (see section 3.2) for maintenance and debugging purposes. Operational logs are TBD.

Changes with respect to IRACE:

No change with respect to IRACE.

14. Real-Time Display Interface

The ***NGC Real-Time Display*** is divided into two parts: the graphical RTD application (see [RD10] and [RD11]) and the underlying data transfer task (DTT) which directly interfaces to the acquisition process.

14.1. Data Transfer Task (ngcrtdDtt)

The data transfer task continuously requests images from a data acquisition process running on the NGC-LLCU and transfers the received images to the RTD application by copying the data into the shared memory and then issuing an image event to the ***rtd-server*** (see [RD10], [RD11] and [RD12]).

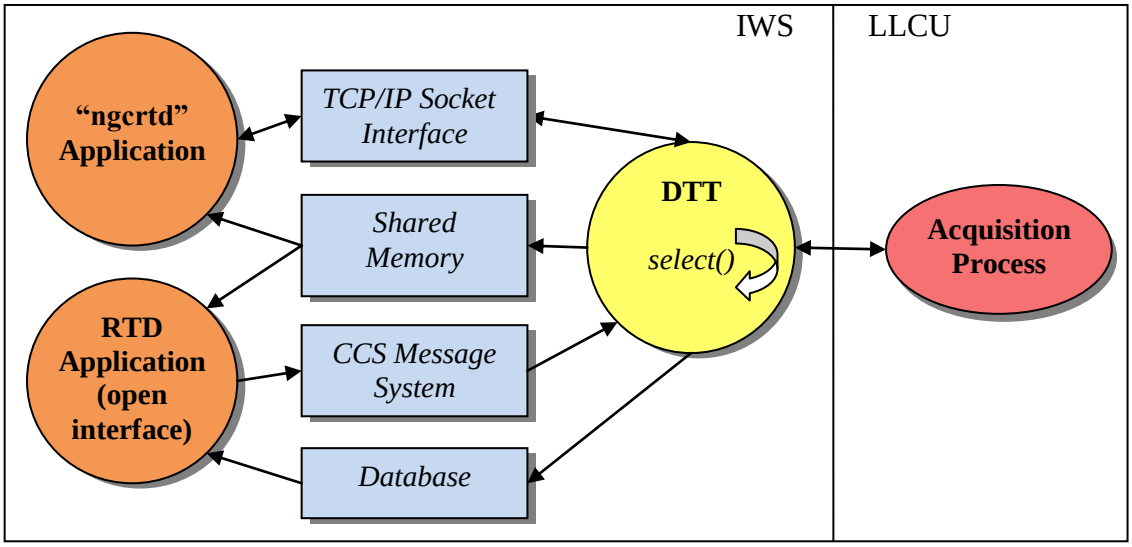

Figure 16 Data Transfer Task

The ***ngcrtdDtt*** data transfer task process is intended to be launched by the RTD application (see command line options in section 14.1.1). It can be controlled via a tiny TCP/IP socket interface. There are only a few control commands to select the frame type to be displayed and to *start/stop* the data-transfer. The *start/stop* command is required to avoid unwanted network load. The data transfer task in turn needs to inform the RTD application about the available frame types in order to properly configure a selection button. Also this frame list can be exchanged via a TCP/IP socket message sent from the data transfer task to the RTD. Additionally the data transfer task has to inform the RTD about its own "*connected*" status (i.e. whether it is actually connected to a data acquisition process or not).

In combination with this ***ngcrtdDtt*** process the RTD application "***ngcrtd***" (see section 14.5) provides a very lightweight real-time display tool which in its basic functionality does not rely on any CCS environment running on the hosting computer.

For more enhanced applications the ***ngcrtdDtt*** data transfer task also includes a CCS-message handler within the data acquisition loop. With this interface it is no longer necessary to launch a first display instance just for having the data transfer task running and issuing image events. By exporting the "*available-frames*" list and the "*connected*"-status to the CCS online database (section 14.1.2) the data transfer task has a standard interface to any CCS application (see Figure 16). The control messages ("*select frame*", "*start/stop*") can be sent to the process via commands defined in a command definition table (CDT, section 14.1.4).

14.1.1. Command Line

The data transfer task is launched with the following command line:

ngcrtdDtt […options…]

Command line options:

-inst <name>	Set instance name. If not specified no instance name is used.
-branch <name>	Database branch name. The default value is "*ngcrtd*".
-host <name>	Host name where the acquisition process is running. A default value for this is given in the ***$NGCPP_HOST*** environment variable.
-port <number>	Data-port number of acquisition process. A default value for this is given in the ***$NGCPP_DATA*** environment variable.
-dcs <name>	DCS instance (default: "").
-mod <index>	DCS acquisition module index. Default is 1.
-q	Use queued transfer. This would not request the "*latest available frame*" but the "*oldest available and not yet transferred frame*" instead. The *–q* option may introduce unwanted display latency and is intended for science data interfaces where the transfer priority is to avoid skipping any valid data. Do not use this option for video-style interfaces.
-noccs	Disable CCS interface.
-camera <name>	Unique RTD camera name. A default value for this is given in the ***$RTD_CAMERA*** environment variable.
-nx <pixels>	Maximum number of pixels in x-direction. Default is 2048.
-ny <pixels>	Maximum number of pixels in y-direction. Default is 2048.
-nbuf <n>	Number of display buffers. Default is 2.
-freq <Hz>	Maximum display frequency (Hz). Default is 0 Hz – no limit.
-verbose <level>	Verbose level. Default is 0.
-log <level>	Logging level. Default is 0.
-rtdPid <pid>	RTD application process id (internal use only).
-rtdPort <number>	RTD application port number (internal use only).
-h or –usage	Show options.

Table 34 Data Transfer Task Command Line Options

14.1.2. Database

A database point has to be created when using the ***ngcrtdDTT*** CCS interface:

```
#define ngcrtdROOT myRoot (e.g. ":Appl_data:myRtd")
#define ngcrtdINSTANCE myName
#define ngcrtdBRANCH myBranch
#include "ngcrtd.db"
```

The *ngcrtd.db* database branch definition file then installs the ***ngcrtdDTT*** class (defined in *ngcrtdDTT.class*) as database point "*myRoot:myBranch*" with the alias name "*<alias>myName:myBranch*".

When ***ngcrtdROOT*** is not defined then the default value "*ngcrtd*" is taken as root.

When ***ngcrtdBRANCH*** is not defined then the branch name is just "*ngcrtd*".

When ***ngcrtdINSTANCE*** is not defined then the ***ngcrtdDTT*** class is installed directly as database point "***ngcrtdROOT***" and the alias is just "***<alias>ngcrtdBRANCH***".

The ***ngcrtdDTT*** class contains the attributes as shown in Table 35.

Attribute	Type	Description
stateName	BYTES32	System status name ("ONLINE"|"OFF").
state	INT32	System status (ccsSTATE_ONLINE| ccsSTATE_OFF).
connected	INT32	Connected to acquisition process (0|1).
enabled	INT32	Data transfer enabled (0|1).
frame	BYTES64	Name of selected frame.
frames	Vector[32] (BYTES64)	List of available frames.
win	Vector[4] (INT32)	Requested window {*start-x, start-y, nx, ny*}.
alarm	BYTES256	Alarm message.

Table 35 Database Class (ngcrtdDTT.class)

14.1.3. Multiple Instances

The instantiation of the *ngcrtdDtt* process is done in the following way: The instance name passed with the "*-inst*" command line option is appended as _"*instance-name*" to the process name when registering with the CCS environment:

"***CCS process name***" = ***ngcrtdDtt_***"***instance-name***"

That way the ***ngcrtdDtt.cdt*** command definition table can be kept for all instances. The database point is by default set to *<alias>*"*instance-name*":*ngcrtd*. It can be changed to *<alias>*"*instance-name*":"*branch-name*" with the "*-branch*" command

line option. With this scheme the *ngcrtd* database class can easily be included as sub-branch in the database of the calling applications (the instance name then matches exactly the instance name of the parent).

When no instance name is given then the default point is just *<alias>ngcrtd* (or *<alias>*"*branch-name*" when the "*-branch*" option is present) and the CCS process name is "*ngcrtdDtt*". When the "-branch" option specifies an absolute path (e.g. "*:Appl_data:myRtd:ngcrtd*") then this value is taken as database branch and the "*-inst*" option just defines the CCS process name. That way the CCS process instance name and the database branch can explicitly be decoupled.

Caution:
*The total length of the "**CCS process name**" is limited to 19 characters. This actually limits the instance name to 9 characters. As in some cases the instance name to be forwarded to the **ngcrtdDtt** process might already be an instantiated display client name (e.g. **ngccrtdGui_myDisp**) the convention is that if the instance name contains an underscore character then the **ngcrtdDtt** process automatically extracts the actual "instance"-part (everything after first underscore) and adds just this to its own process name in order to have a reasonable length (e.g. **ngcrtdDtt_myDisp**).*

14.1.4. Commands

The following commands are defined in the command definition table (***ngcrtdDtt.cdt***).

BREAK	Interrupt server (SIGUSR1).		
CONFIG	**-camera**	String	Set new camera name.
	-host	String	Acquisition process host name.
	-port	Integer	Acquisition process data port number.
	-nx	Integer	Maximum x-dimension.
	-ny	Integer	Maximum y-dimension.
	-nbuf	Integer	Number of display buffers.
	-freq	Real	Maximum display frequency (Hz). A value less than or equal to zero indicates that there is no limit applied.
EXIT	Exit server (go to ***OFF*** state).		
FRAME	**-select**	String	Select frame.
	-win	Integer	Requested window – repeated four times: {*start-x, start-y, nx, ny*}.
	-id	Integer	Image id for rtdServer image-event (normally 0).
	-retransmit	Logical	Retransmit current frame..
KILL	Send a KILL signal to the server (SIGUSR2). The data transfer task will go to ***OFF*** state.		
INVERT	Invert current image.		
MSGDLOG	Disable auto-logging of messages sent or received by the application.		
MSGELOG	Enable auto-logging of messages sent or received by the application.		
SETUP	**-expoId**	Integer	Ignored.
	-file	String	Load setup from a file. Unless an absolute path is given, the setup file is searched in the directory `$INS_ROOT/$INS_USER/COMMON/SETUPFILES/DET`
	-function	String	Pairs of "keyword" "value". Can be repeated to set multiple setup keywords.
	-default	Logical	Apply default setup.
START	Start data transfer from acquisition process to data transfer task.		
STATUS	**-expoId**	Integer	Ignored.
	-function	String	Retrieve the status of all given keywords. Can be repeated. If no function is given then a list of all parameters is returned.
STATE	Return actual data transfer task status (ONLINE|OFF).		
STOP	Stop data transfer from acquisition process to data transfer task.		
VERBOSE	**-on**	Logical	Switch generation of verbose messages on.
	-off	Logical	Switch generation of verbose messages off.
VERSION	Returns the actual server version (ASCII-string).		

Table 36 Data Transfer Task Command Interface

14.1.5. SETUP and STATUS

The ***SETUP*** and ***STATUS*** commands can be used to configure/query the data transfer task via keywords. The following keywords have been defined:

Keyword	Type	Description
DET.RTD.FRAME	String	Select frame type to be displayed.
DET.RTD.CAMERA	String	Camera name.
DET.RTD.NX	Integer	Maximum x-dimension.
DET.RTD.NY	Integer	Maximum y-dimension.
DET.RTD.ACQHOST	String	Acquisition process host name.
DET.RTD.ACQPORT	Integer	Acquisition process data port number.
DET.RTD.INVERT	Logical	Invert real-time image data.
DET.RTD.WIN.STRX	Integer	Window request – lower left x.
DET.RTD.WIN.STRY	Integer	Window request – lower left y.
DET.RTD.WIN.NX	Integer	Window request – size in x-direction.
DET.RTD.WIN.NY	Integer	Window request – size in y-direction.
DET.RTD.BUFNUM	Integer	Number of display buffers.
DET.RTD.MAXFREQ	Double	Maximum display frequency (Hz). A value less than or equal to zero indicates that the display frequency has no limit. The minimum display frequency is 0.1 Hz.

Table 37 Setup and Status Keywords (ngcrtdDtt)

Keyword	Type	Description
DET.RTD.CONNECT	Logical	Data transfer task is connected to an acquisition process.
DET.RTD.ENABLED	Logical	Data transfer from acquisition process to data transfer task is enabled.
DET.RTD.STATE	String	Actual status of the data transfer task process (ONLINE|OFF).

Table 38 Status Keywords (ngcrtdDtt)

14.1.6. How to create a new NGC RTD interface

This section addresses to software engineers and presupposes knowledge of the C++ programming language and of the ESO VLTSW environment.

A new display interface is created by deriving a new class from the NGC RTD display interface class ***ngcrtdRTD***:

```
class xxrtdDISP: public ngcrtdRTD
{
public:
  // Constructor(s)
  xxrtdDISP(){
    // We assign our own name
    Type("XXRTD");
  }

  // Destructor
  virtual ~xxrtdDISP(){}

  // Display call-back
  int Display(int idx) {
    /*
     * Here we may add some post-processing on
     * frame[idx].memPtr and frame[idx].hdr
     */

    // We use the standard display
    return (ngcrtdRTD::Display(idx));
  }

protected:

private:
};
```

Then a main process has to be created according to the following template (this is based on the "*ngcrtd/templates/xxrtdDtt.C*" program). The process name can then be passed to the ngcrtd application with the "*-server <process name>*" command line option (see Table 42).

```
/* Control server process */
static int serverProcess(int argc, char **argv)
{
  xxrtdDISP display;            // our display
  ngcrtdDTT server(&display);   // the server with our display
  int exitStatus = 0;

  // Parse command line arguments
  if (server.ParseArguments(argc, argv) == ngcbFAILURE)
    {
    if (strlen(server.ErrMsg()) > 0)
      {
      fprintf(stderr, "xxrtdDtt: %s\n", server.ErrMsg()); return (1);
      }
    else
      {
      server.PrintUsage(argv[0]); return (0);
      }
    }

  // Initialize
  if (server.Initialize() == ngcbFAILURE)
    {
    fprintf(stderr, "xxrtdDtt: %s\n", server.ErrMsg()); return (1);
    }

  // Main loop
  exitStatus = server.MainLoop();
  if (exitStatus)
    {
    fprintf(stderr, "xxrtdDtt: %s\n", server.ErrMsg()); return (1);
    }
  return (exitStatus);
}

/* Main entry point */
int main(int argc, char *argv[])
{
  int exitStatus;

  // Call server
  exitStatus = serverProcess(argc, argv);
  exit(exitStatus);
}
```

This example together with an appropriate *Makefile* is provided as template in the ***ngcrtd*** software module:

```
ngcrtd/templates/xxrtd
```

The ***xxrtd*** template is installed in:

```
$INTROOT/templates/forNGC/
$VLTROOT/templates/forNGC/
```

Changes with respect to IRACE:

The interface to the acquisition process is backwards compatible to IRACE.

14.2. Control Mega-Widget

A mega-widget for controlling the data transfer task has been created to be embedded into application GUIs:

Figure 17 Control Mega-Widget

The widget can be added by pressing the "*Import Mega Widget…*" button in the ***panelEditor*** [RD8]. The widget is located in the *libngcrtdPan.tcl* library:

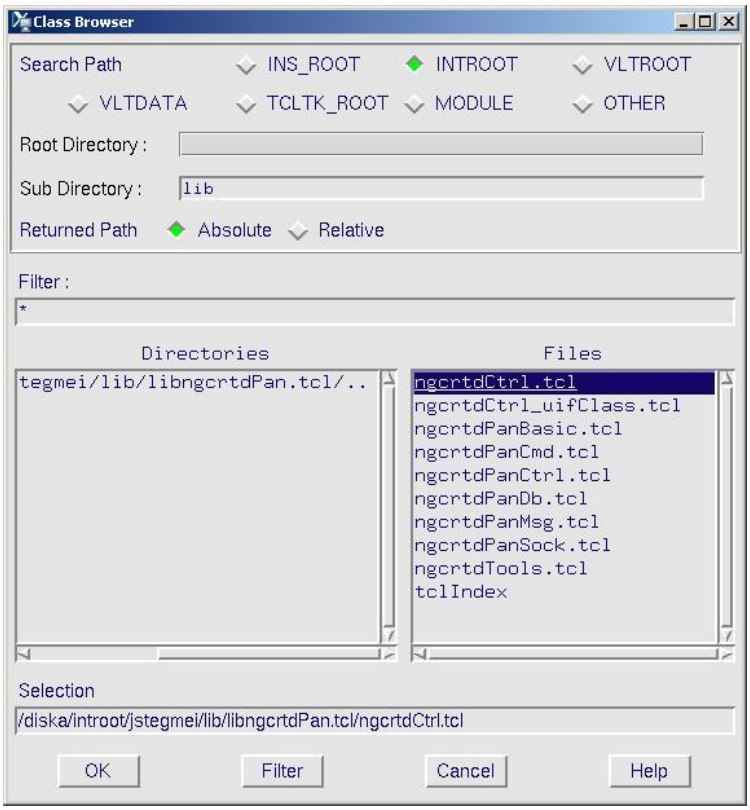

Figure 18 Importing the Mega-Widget

14.2.1. Widget Options

Name	Type	Description	Default
-open	(0|1)	When set to 1 the data transfer task connection is opened automatically when the widget is created. The widget is configured with default values or via the command line arguments (if present). When set to 0 the connection has to be established with the *openDtt{}* method. This allows a prior re-configuration of the widget without having the data transfer task process re-allocate any resources.	1
-server	String	Data transfer task process name.	"ngcrtdDtt"
-dbcwp	String	Data transfer task database working point. When an <*alias*> is given then the data transfer task process instance becomes the working point without the <*alias*>,	""
-dbcwpVar	String	Name of global variable defining the DTT database working point. An exception is the string "CWP" in which case the actual value of the global variable *$::cwp* is used. Only one of *–dbcwp* and *–dbcwpVar* should be specified.	""
-inst	String	Data transfer task process instance. This option should only be specified when the "*-dbcwp*" option does not use an <*alias*>.	""
-branch	String	Data transfer task database branch name.	"ngcrtd"
-camera	String	Camera name.	$RTD_CAMERA
-dbdcs	String	DCS database point.	""
-host	String	Acquisition process host name.	$NGCPP_HOST
-dataport	Integer	Acquisition process data port number.	$NGCPP_DATA
-dcs	String	DCS instance.	""
-mod	Integer	DCS acquisition module index.	1
-dim	String	Maximum camera dimension. The format is a geometry string (e.g.: "*1024x1024*")	"2048x2048"
-nbuf	Integer	Number of display buffers.	2
-freq	Real	Maximum display frequency (Hz).	no limit
-fpclr	Script	Call-back script to clear a fix. pattern upon various events (e.g. changes in size or type).	""
-verbose	Integer	Verbose level.	0
-socket	(0|1)	Use socket interface.	0
-rtd	String	RTD interface type. Can be one of: *rtd, rtd0, rtdb, rtdc, native*	"native"

Table 39 Widget Options

In addition the standard options inherited from the ***UifPanel*** class (see [RD8]) are available.

When the options are left empty a default assignment is done by parsing the command line options as stored in the global *argv* array variable. When the *argv* array variable does not exist or an option is not given therein then environment variables (***$NGCPP_HOST*** for "*-host*", ***$NGCPP_DATA*** for "*-dataport*" and ***$RTD_CAMERA*** for "*-camera*") are used. The other options employ hardcoded default values in that case.

Caution:
Once the data transfer task process has been launched the widget configuration options may imply a re-allocation of the image resources (e.g. shared memory) or even a re-start of the data transfer task process.

When pressing the right mouse button within the widget canvas a control menu is raised. The menu enables the user to reset the underlying data transfer task and also to call a properties dialog (see Figure 20). The properties dialog allows the user to reconfigure various resources and to connect to a different image data server (acquisition process).

14.3. Widget Methods

The public method "*openDtt {}*" opens the connection to the NGC data transfer task – i.e. launches and configures the data transfer task process. Depending on the *"-open*" widget option this is automatically done when the widget is created.

The public method "*closeDtt {}*" terminates the data transfer task process and closes the connection.

14.3.1. Widget Instantiation

The "*-dbcwp*" option is used to assign a unique instance to the widget. When this is given as database alias name (e.g. "*<alias>myGui*") then the instance name of the data transfer task process is set to the option value without the <*alias*>:

```
Widget options:  -dbcwp <alias>myGui -branch myBranch
CCS process name:"ngcrtdDtt_myGui"
Database branch: "<alias>myGui:myBranch"
```

When the "*-dbcwp*" option specifies an explicit branch (e.g. "*:Appl_data:myRoot*") then the data transfer task process instance name cannot be derived from this option. In that case the instance name has to be set explicitly with the "*-inst*" option:

```
Widget options:  -dbcwp :Appl_data:myRoot -inst myGui -branch myBranch
CCS process name:"ngcrtdDtt_myGui"
Database branch: ":Appl_data:myRoot:myBranch"
```

When only the "*-inst*" option is given then the database working point becomes the option value preceded by the term <*alias*>:

```
Widget options:  -inst myGui -branch myBranch
CCS process name:"ngcrtdDtt_myGui"
Database branch: "<alias>myGui:myBranch"
```

The *dbcwp* value may also be specified via the name of a global variable (*-dbcwpVar <variable-name>*, see Table 39). In this case the *dbcwp* is set to *$gvar(<variable-name>).* An exception is the string "*CWP*" in which case the actual value of the global variable *$::cwp* is used. Only one out of these two options must be specified when creating the widget.

14.4. NGC RTD Application (ngcrtdGui)

The ***ngcrtd*** software module provides an executable display application which is based on the ***rtdc*** framework [RD11]. The application is started with the following command line:

```
ngcrtdGui […options…]
```

The application takes care of launching/terminating the data transfer task process.

14.4.1. Command Line Options

-ccsProcName <name>	Name used to register ***ngcrtdGui*** with the CCS message system. The default name is ***ngcrtdGui_pid*** (*pid* is the process id of the application).
-camera <name>	Unique RTD camera name. A default value for this is given in the ***$RTD_CAMERA*** environment variable.
-attach <0|1>	Attach camera (default: 1).
-enable	Enable transfer.
-invert	Invert real-time image.
-dbcwp <name>	RTD database working point (default: "<alias>ngcrtdGui") – see [RD11].
-inst <name>	RTD instance name (default: ngcrtdGui) – see [RD11].
-dtt <name>	Data transfer task instance name (default: ngcrtdGui).
-server <name>	Data transfer task server name.
-branch <name>	Data transfer task database branch name.
-noccs	Disable CCS message system interface.
-host <name>	Host name where the acquisition process is running. A default value for this is given in the ***$NGCPP_HOST*** environment variable.
-dataport <number>	Data-port number of acquisition process. A default value for this is given in the ***$NGCPP_DATA*** environment variable.
-dcs [name]	DCS instance (default: "").
-mod <index>	DCS acquisition module index. Default is 1.
-nx <pixels>	Maximum number of pixels in x-direction.
-ny <pixels>	Maximum number of pixels in y-direction.
-dim <geometry>	Same as nx/ny but with geometry syntax.
-nbuf <n>	Number of display buffers. Default is 2.
-freq <Hz>	Maximum display frequency (Hz). Default is 0 Hz – no limit.
-dbdcs <name>	NGC DCS database root (default is "<*alias*>*ngcdcs*").
-verbose <level>	Verbose level. The default value is 0.
-usage	Show options.

Table 40 Command Line Options (ngcrtdGui)

The ***ngcrtdGui*** process also uses by default ***$NGCPP_HOST*** for the hostname (or ***$HOST*** if ***$NGCPP_HOST*** is not set) and ***$NGCPP_DATA*** for the port-number. These values can be overwritten with the *–host <name>* and *–dataport <number>* command line options (see Table 40).

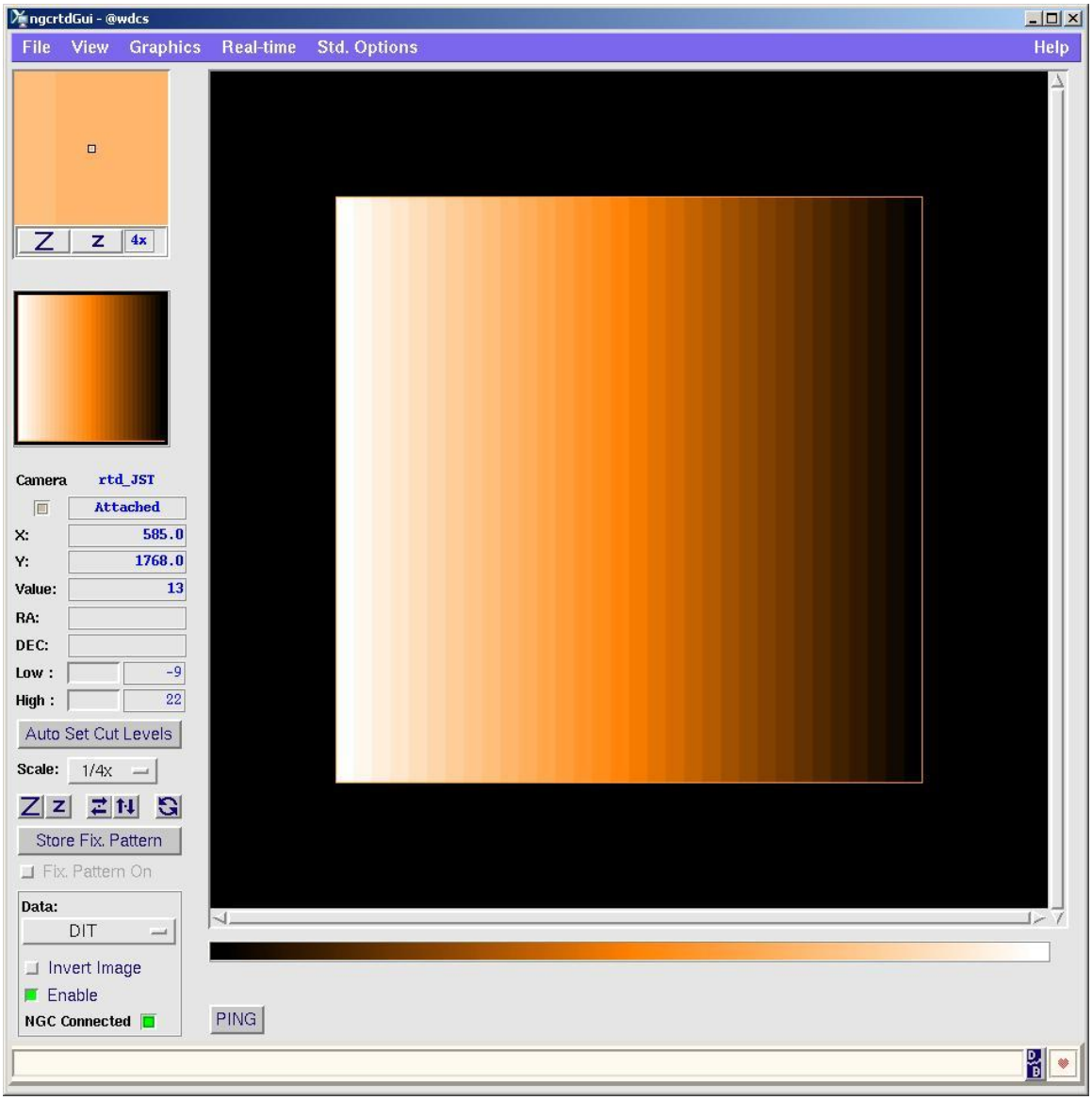

Figure 19 ngcrtdGui

14.4.2. Application Database

The trivial way is to include the following lines in the ***DATABASE.db*** file of the CCS environment:

```
#include "ngcrtdApp.db"
```

And then just use the default instance and database options to start the ***ngcrtdGui***.

Further instances can be created in the following way:

```
#define ngcrtdROOT <myROOT> (e.g. ":Appl_data:myApps")
#define ngcrtdAPP myRtd
#include "ngcrtdApp.db"
```

This creates an ***rtdc*** instance

```
<myRoot>:myRtd (e.g. ":Appl_data:myApps:myRtd")
```

with the alias

```
<alias>myRtd
```

and adds the ***ngcrtdDTT*** class at

```
myRoot:myRtd:ngcrtd (e.g. ":Appl_data:myApps:myRtd:ngcrtd")
```

Then the display application can be launched with the command:

```
ngcrtdGui -inst myRtd
```

When ***ngcrtdROOT*** is not defined the default value "***:Appl_data***" is used. When ***ngcrtdAPP*** is not defined the default value "***ngcrtdGui***" is used.

The same can be achieved by defining the optional ***ngcirconRTD*** macro before including *ngcircon.db* (see section 6).

14.4.3. RTD Command Interface

The ***ngcrtdGui*** application registers with the CCS message system using the name ***ngcrtdGui_pid*** where *pid* is the process id of the application. To use another CCS registration name the application has to be started with

```
ngcrtdGui -ccsProcName myName
```

Then the RTD functions can be accessed through the ***SCRIPT*** command:

```
msgSend -n "" myName "SCRIPT" "…code…"
```

The following short-cut codes are defined to call the ***ngcrtdGui*** specific control functions:

`ngcrtdGuiCamera {name}`	Set new camera name.
`ngcrtdGuiAttach {0|1}`	Attach/detach camera.
`ngcrtdGuiInvert {0|1}`	Invert image.
`ngcrtdGuiEnable`	Start displaying images.
`ngcrtdGuiDisable`	Stop displaying images.
`ngcrtdGuiSelect {name}`	Select frame-type to be displayed by name (DIT, INT, …)
`ngcrtdGuiWin {sx sy nx ny}`	Select window (soft window).
`ngcrtdGuiRetransmit`	Retransmit current image.
`ngcrtdGuiFpStore`	Store current image as fix.-pattern.
`ngcrtdGuiFpClear`	Clear fix.-pattern.
`ngcrtdGuiFpOn`	Enable fix.-pattern subtraction
`ngcrtdGuiFpOff`	Disable fix.-pattern subtraction
`ngcrtdGuiRotate {0|1}`	Rotate image.
`ngcrtdGuiFlip {x|y 0|1}`	Flip image in x or y direction.
`ngcrtdGuiClear`	Clear current image.
`ngcrtdGuiAutoCut`	Apply automatic cut levels.

Table 41 ngcrtdGui SCRIPT Commands

The ***SCRIPT***-functions provided by the ***rtdc*** framework are described in [RD11].

14.4.4. Properties

The control widget allows the user to interactively reconfigure the data transfer properties:

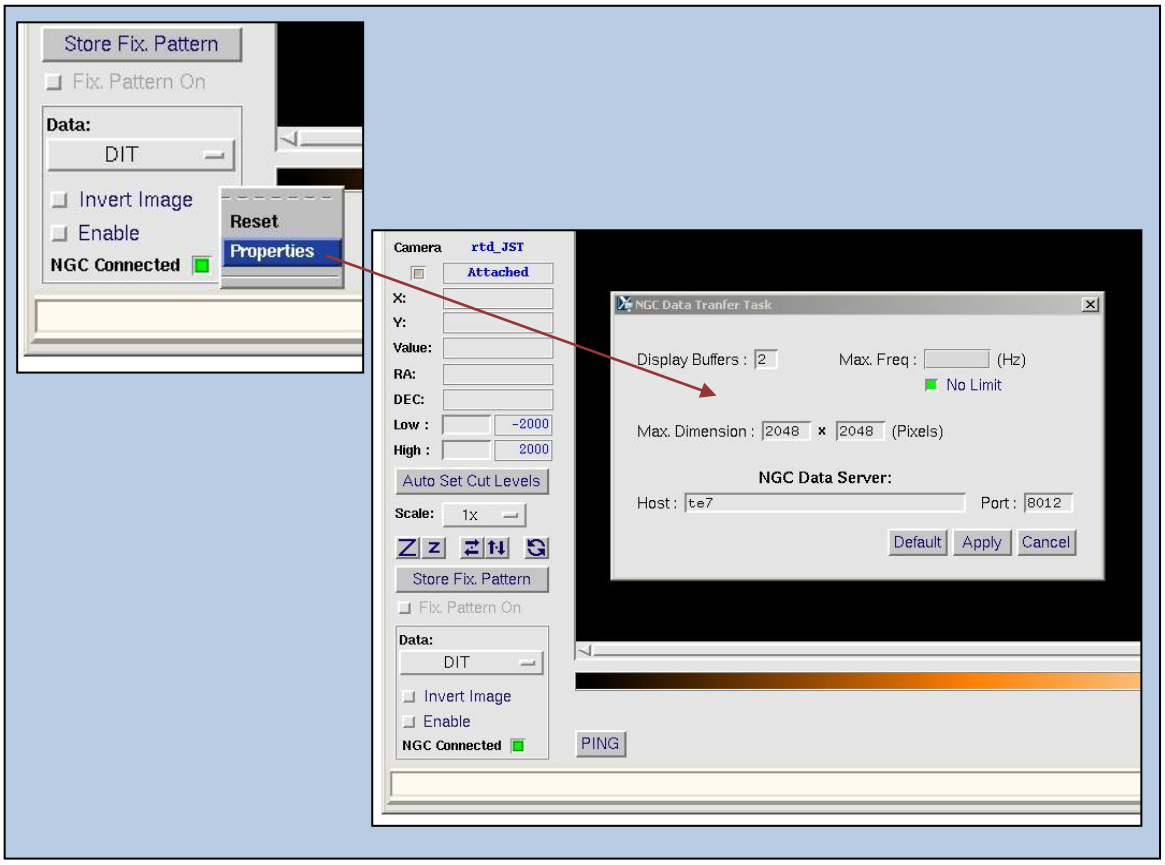

Figure 20 ngcrtdGui Control Menu & Properties

14.4.5. The NGC Data-Cube Player

The ***ngcrtdGui*** application has the capability to playback multi-dimensional FITS-files (data-cubes, NAXIS = 3). The player (***ngcbCube2Rtd***, see section 27.6) is launched from the menu and can either be operated manually or automatically plays the image slices at selectable speed.

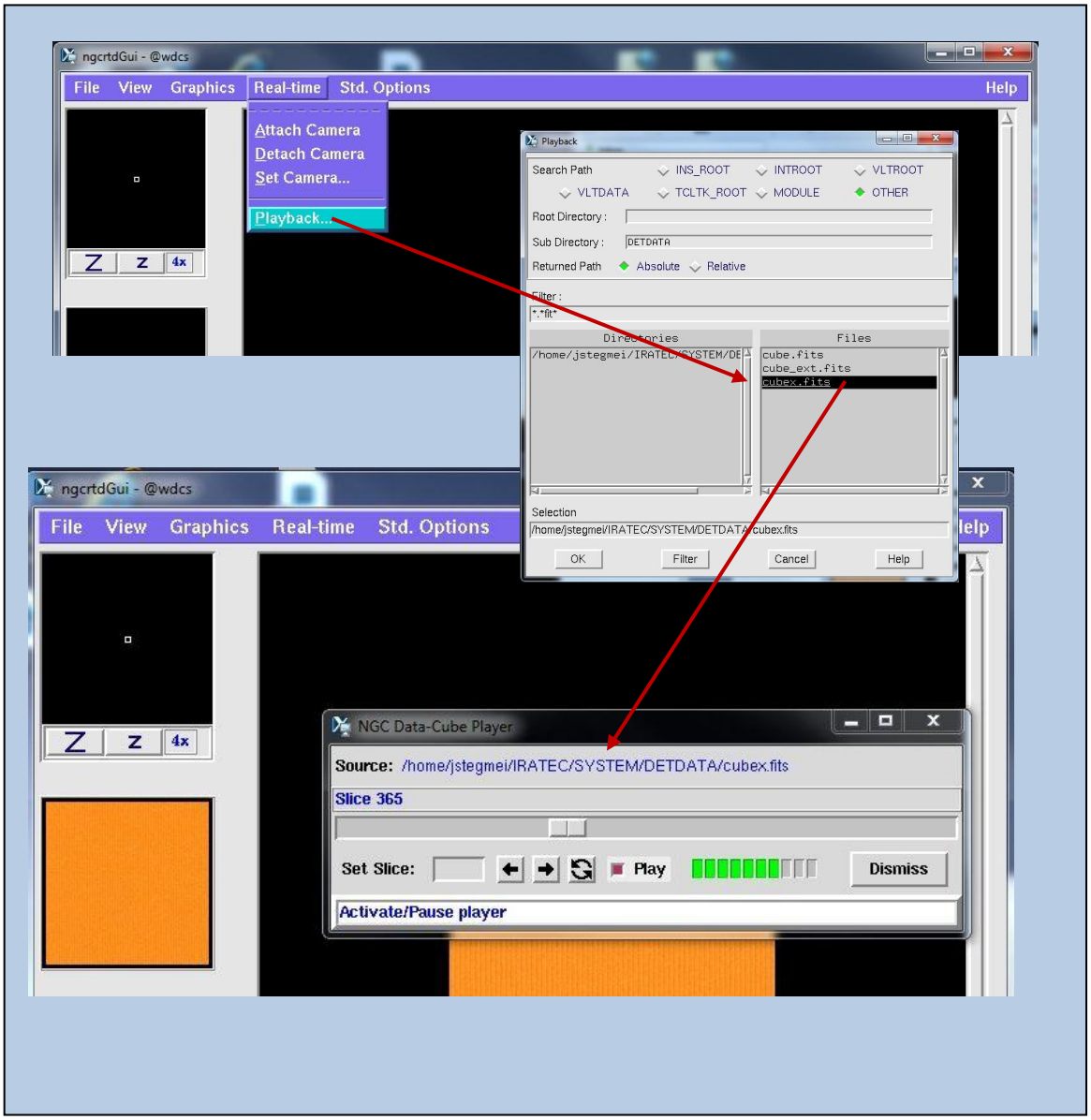

Figure 21 NGC Data-Cube Player

14.5. NGC RTD Application (ngcrtd)

Caution:

*The **ngcrtd** application has been declared discarded with the VLT2011 software release. The application requires the additional software module "**rtdb**" to be installed which is no longer part of the official VLT software releases. Since VLT2013 the framework can no more be used at all. New applications should make use of the **rtdc** framework [RD11] and plug in the NGC control widget as mentioned in the previous chapters or use the **ngcrtdGui** application as described in section 14.4.*

*Since VLT2013 the **ngcrtd** script just checks the CCS-environment and then launches the **ngcrtdGui**. If the environment is not yet running then it is automatically started. If it does not yet exist then a minimum default CCS-environment is created first.*

The ***ngcrtd*** application is started with the following command line:

```
ngcrtd […options…]
```

The application takes care of launching/terminating the data transfer task process.

Command line options:

-camera <name>	Unique RTD camera name. A default value for this is given in the ***$RTD_CAMERA*** environment variable.
-server <name>	Data transfer task server name.
-host <name>	Host name where the acquisition process is running. A default value for this is given in the ***$NGCPP_HOST*** environment variable.
-dataport <number>	Data-port number of acquisition process. A default value for this is given in the ***$NGCPP_DATA*** environment variable.
-dcs [name]	DCS instance (default: "").
-mod <index>	DCS acquisition module index. Default is 1.
-nx <pixels>	Maximum number of pixels in x-direction.
-ny <pixels>	Maximum number of pixels in y-direction.
-nbuf <n>	Number of display buffers. Default is 2.
-freq <Hz>	Maximum display frequency (Hz). Default is 0 Hz – no limit.
-appname <name>	Unique application name.
-ccs <T|F>	Enforce/suppress data transfer task CCS registration.
-dbdcs <name>	NGC DCS database root (default is "<*alias*>*ngcdcs*").
-verbose <level>	Verbose level. The default value is 0.
-usage	Show options.

Table 42 Command Line Options (ngcrtd)

The ***ngcrtd*** process by default uses ***$NGCPP_HOST*** for the hostname (or ***$HOST*** if ***$NGCPP_HOST*** is not set) and ***$NGCPP_DATA*** for the port-number. These values can be overwritten with the *–host <name>* and *–dataport <number>* command line options (see Table 42).

When the acquisition process is launched by the control server it gets the data-port number as specified in the system configuration file with the ***DET.ACQi.DATAPORT***

keyword (see section 17.1.3). This is normally set to ***$NGCPP_DATA*** but may be any other environment variable or also a hard-coded value. When the acquisition process is started manually then it uses ***$NGCPP_DATA*** by default. This can be overwritten with the *-dataport <number>* acquisition process command line option.

It is possible to control the "*start/stop displaying*" via commands:

```
msgSend -n "" ngcrtd "SCRIPT" "ngcrtdStart"
msgSend -n "" ngcrtd "SCRIPT" "ngcrtdStop"
```

This is the same as clicking the "*Start*" or "*Stop*" button. In order to send this command via ***msgSend*** one has to set the keyword:

```
RTDB.CCSINIT T;
```

in the configuration file:

```
$INS_ROOT/SYSTEM/COMMON/CONFIGFILES/ngcrtdRTDB.cfg
```

It is possible to run several ***ngcrtd*** instances in parallel:

```
ngcrtd -camera CAM-1 -appname APP-1 -host <acq.-proc host 1>
       -dataport <acq.-proc data port 1>

ngcrtd -camera CAM-2 -appname APP-2 -host <acq.-proc host 2>
       -dataport <acq.-proc data port 2>

            :

ngcrtd -camera CAM-N -appname APP-N -host <acq.-proc host N>
       -dataport <acq.-proc data port N>
```

The value of "***CAM-1***" … "***CAM-N***" and "***APP-1***" … "***APP-N***" can be any significant string.

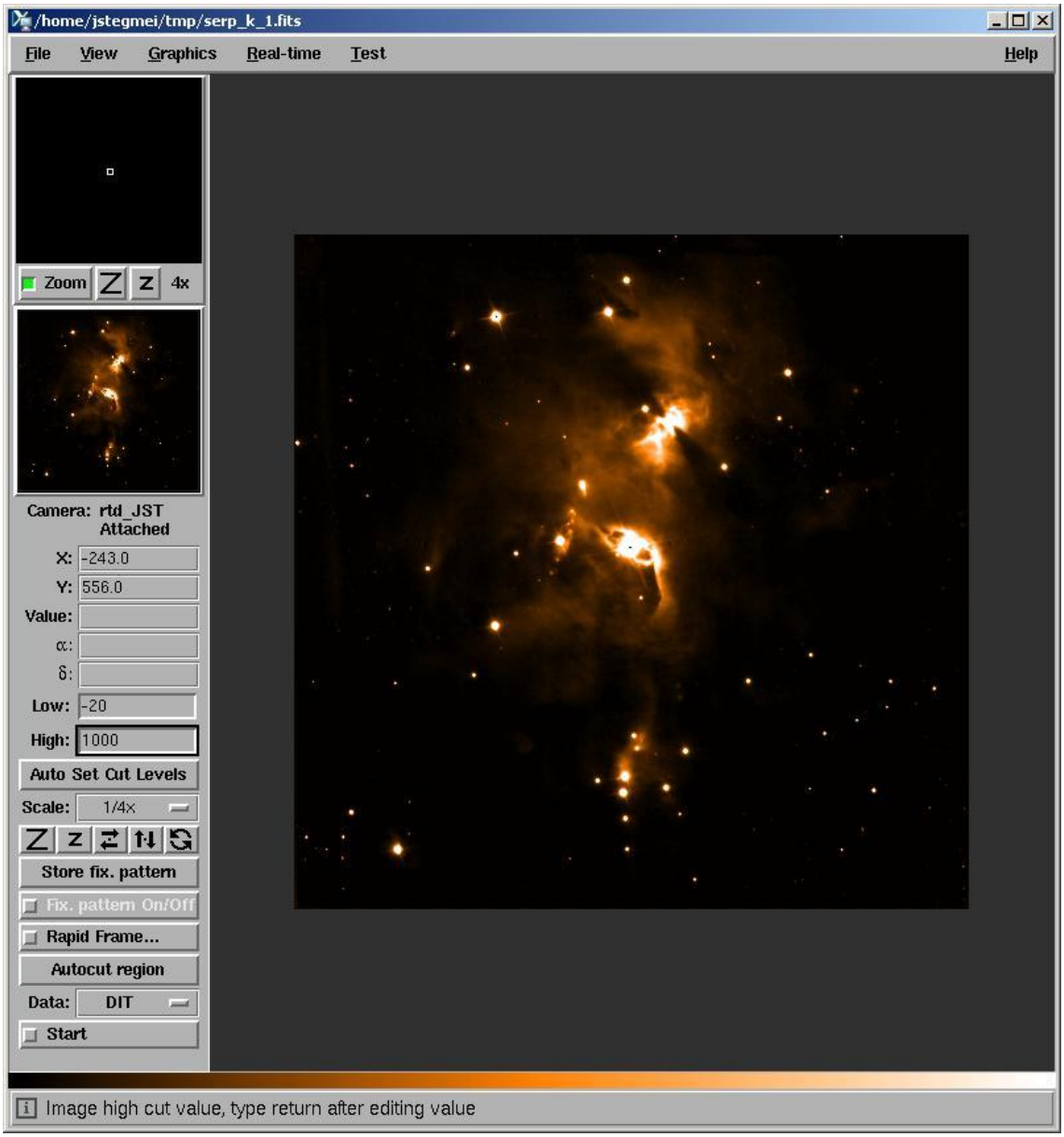

Figure 22 ngcrtd

15. Graphical User Interface

Graphical user interfaces are available for system control (***ngcgui***) and for editing the controller electronics system configuration (***ngcguiCfg***). The GUIs are part of the ***ngcgui*** software module. The graphical user interface communicates with the control server via the online database (see section 6) and the command interface (see section 4) provided by the CCS message system.

15.1. Engineering GUI

The engineering GUI is launched with the following command line:

```
ngcgui [options]
```

Command line options:

-db <point>	Database point to be used (without the instance label and without the <alias>). If this option is not specified, the default database point '***<alias>ngcdcs***' will be used. The instance label (if given) will always be appended to the point-name.
-inst <label>	Instance label. By default no instance label is used.
-server <name>	Control server name (without instance). Default is "*ngcdcsEvh*".
-restart	(Re-)start control server after GUI has been launched.
-cfg <file-name>	Set a default controller electronics system configuration-file (see Table 3).
-ld <dictionary>	Load the given dictionary. The option may be repeated to load more than one additional dictionary. The common dictionary "*ESO-VLT-DIC.NGCDCS*" is always loaded into the system and needs not to be specified using this command line option.
-xterm	Start all (sub-) processes in a separate terminal. This only takes effect, when the verbose level is greater than zero.
-verbose <level>	Verbose level (default is 0).
-log <level>	Logging level (default is 0).
-silent	Do not show startup progress panel.
-usage	Show available command line options.

Table 43 Engineering GUI Command-Line Options

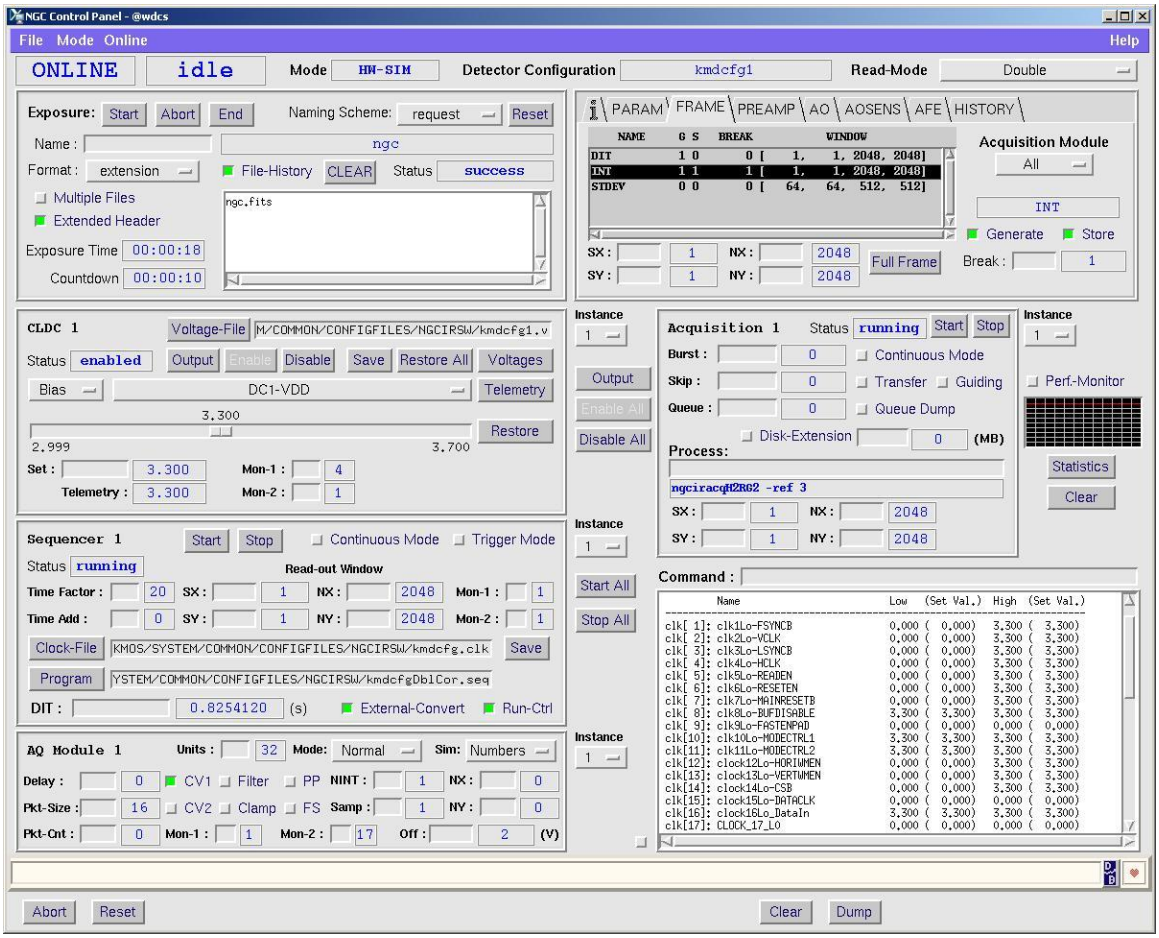

Figure 23 Engineering GUI

Caution:

The number of database event attachments is limited. When the panel attaches to the attributes of the maximum possible number of DFE modules (24) then the resources are exhausted when launching a second GUI. A workaround is to limit the number of attached modules to the maximum expected for a certain installation/instrument. This can be done by setting the ***DET.CON.MAXMOD*** keyword in the startup configuration (see Table 4) or by writing directly to the online database:

Example:

```
> dbWrite <alias>ngcircon[_<instance>].maxMod 4
```

This instructs the GUI to attach only to 4 DFE modules.

15.2. Hardware Control GUI

For pure hardware control without data acquisition and exposure handling (see section 17.3.1) a reduced GUI is available:

```
ngcguiHw [options]
```

The command line options are the same as for the engineering GUI (see Table 43).

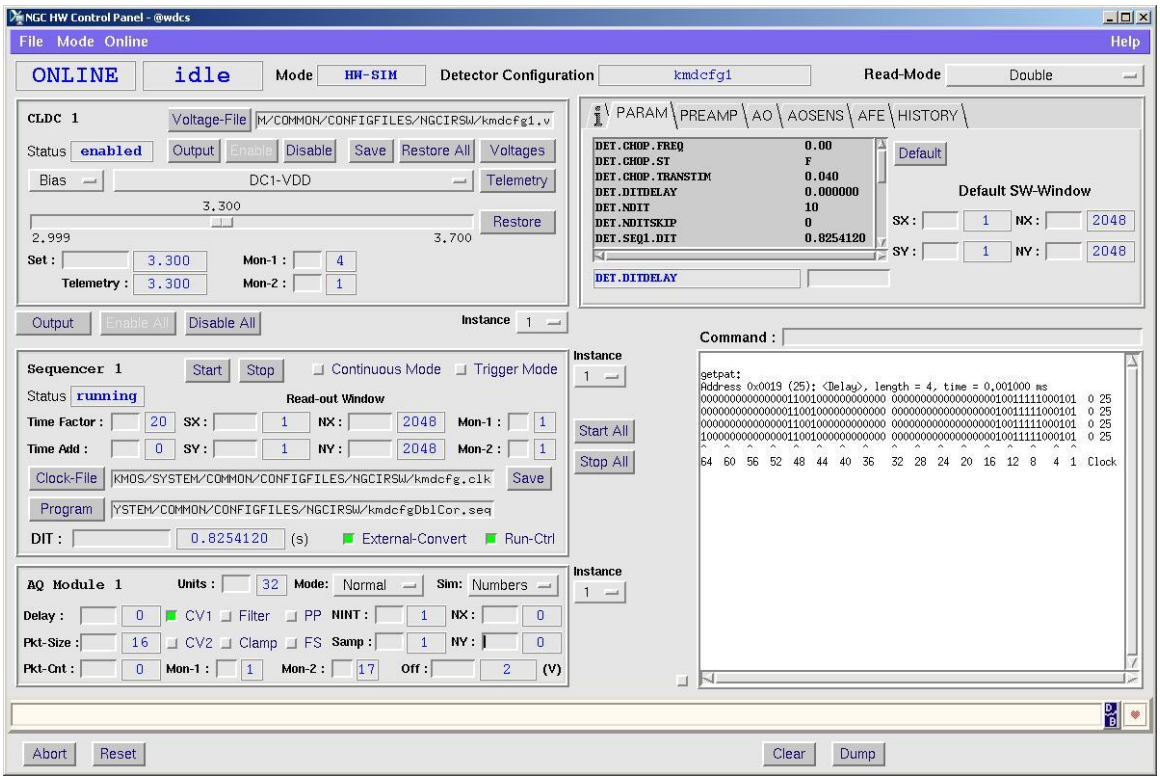

Figure 24 Hardware Control GUI

16. Controller Programming

The controller programming consists of the definition of clock patterns, sequencer programs and the voltage setup. A graphical editing tool (***BlueWave***) is available to generate the described configuration- and program-files.

16.1. Clock Pattern Definition

The clock patterns define sequences of clock states, which are stored in a RAM inside NGC sequencer HW. Those sequences can be executed in a programmable order defined by the sequencer program (see section 16.2). The duration of each pattern state (dwell time) is defined in the state itself. The value is given in ticks. A flag (modification flag) is given for each state to allow/prevent the modification of the dwell time for this state by a global factor (setup keyword ***DET.SEQi.TIMEFAC***).

The sequencer runs at a clock speed of 100 MHz (1 tick = 10 ns). So the dwell time of each clock state can be specified in steps of 10 ns. The minimum pattern length is 2 states. The minimum state length is 2 ticks. Special applications may use a faster or slower sequencer clock. To have the pattern execution times correctly calculated by the software one has to specify the actual sequencer clock-speed in the controller electronics system configuration file. This is only needed when not using the default system. E.g.:

```
DET.SEQ1.CLOCK  5.0E-9;  # 5 ns tick
```

There are two uploading methods for clock pattern definitions. The classification is done via filename extensions ("*.clk*", "*.bclk*"):

- A low-level binary structured format (".bclk") referring to the sequencer pattern RAM directly. This allows complete clock patterns to be produced by an external tool in case the output format of this tool can be adapted to our needs. Here the aim is to have the simplest possible format (still readable but not intended to be edited by hand).
- A more user-friendly ASCII description (short FITS format, ".clk"), which can be edited directly by hand. Here the aim is to have the best readable format.

The control server allows converting from the one into the other format using the command:

"*SEQ -save <filename>*"

Again the filename extension ("*.clk*", "*.bclk*") is used to determine the format.

16.1.1. Pattern Initialization

By default the sequencer sets all clock-states to zero (low value) when the program is reset or (re-)started. This overall default initialization state can be changed with the

DET.SEQi.INITPAT keyword (binary string with up to 64 characters 0|1) or via the ***DET.SEQi.INITLOW/HIGH*** keywords (HEX-values of the pattern RAM low and high words). Additionally the actual initialization value can be set in the clock pattern configuration files as described in the following sections. The life time of the value is then the life time of the clock pattern configuration file (i.e. when a new file is loaded and no value is specified therein then the default initialization value is restored).

16.1.2. Binary Clock Pattern Definition

```
# <INIT>
0
0b00000000000000000000000000000000 0b00000000000000000000000000000010 0
!

# <Pattern-1>
1
0b00000000000000000001000000000000 0b00000000000000000000000100000000 1
0b00000000000000000001000000000000 0b00000000000000000000000100000000 1
0b00000000000000000001000000000000 0b00000000000000000000000000000001 0
0b00000000000000000001000000000000 0b00000000000000000000000000000001 0
0b00000000000000000001000000000000 0b00000000000000000000000100000000 0
0b10000000000000000001000000000000 0b00000000000000000000000100000000 1
!

# <Pattern-3>
3
0b00000000000000000011000000000000 0b00000000000000000000000000000010 1
0b00000000000000000011000000000000 0b00000000000000000000000000000010 1
0b00000000000000000011000000000000 0b00000000000000000000000000000110 1
0b00000000000000000011000000000000 0b00000000000000000000000000000110 0
0b00000000000000000011000000000000 0b00000000000000000000000000000010 0
0b10000000000000000011000000000000 0b00000000000000000000000000000010 1
!
#           High Word               |          Low Word                  |Mod.-
Flag
```

Here the time scale goes vertically from the top to the bottom of each pattern. Each row defines exactly one state. The dwell-time bits [12 – 27] in the high-word define the default dwell-time which again may be modified by a global factor in case the modification flag (third word of each row) is ‘1’. The ‘!’ character marks the end of each pattern. Each block starts with a pattern-number followed by a new-line character. A zero pattern number indicates the initialization pattern to be applied whenever the sequencer program execution is reset (i.e. whenever the sequencer is restarted). This pattern must have only one state (the modification flag must be present but is ignored). When the zero-pattern is omitted then the sequencer uses its default state when being reset/restarted as described before. Basically this format is a direct mirror of the clock-pattern RAM.

16.1.3. ASCII Clock Pattern Definition

```
###############################################################################
# E.S.O. - VLT project
#
# "@(#) $Id$"
#
# who       when        what
# --------  ----------  ----------------------------------------------
# jstegmei  2005-09-05  created
#
###############################################################################
#
# DESCRIPTION
# Example ASCII FILE for NGC sequencer clock pattern definitions.
#
###############################################################################

# Clock mapping (can be spread over several lines).
# This maps the clocks described below onto physical clock lines.
# Mechanism is: Phys. clock line for logical clock n = MAP[n].
DET.CLK.MAP1  "1,2,33";     # Mapping list
DET.CLK.MAP2  "37,4";       # Mapping list

# Optional initialization pattern (applied after sequencer program reset)
DET.PAT.INIT1  0;
DET.PAT.INIT2  1;
DET.PAT.INIT3  0;     # Convert
DET.PAT.INIT4  0;     # Start pulse
DET.PAT.INIT5  0;

# Clock pattern definitions
DET.PAT1.NAME  "Delay";
DET.PAT1.NSTAT 5;
DET.PAT1.CLK1  "00000";
DET.PAT1.CLK2  "11111";
DET.PAT1.CLK3  "00000";     # Convert
DET.PAT1.CLK4  "00000";     # Start pulse
DET.PAT1.CLK5  "00000";
DET.PAT1.DTV   "2,2,2,2,2"; # Dwell-Time vector
DET.PAT1.DTM   "0,0,0,0,0"; # Dwell-Time modification flags

DET.PAT2.NAME  "FrameStart";
DET.PAT2.NSTAT 4;
DET.PAT2.CLK1  "0000";
DET.PAT2.CLK2  "1111";
DET.PAT2.CLK3  "0000";      # Convert
DET.PAT2.CLK4  "0110";      # Start pulse
DET.PAT2.CLK5  "0000";
DET.PAT2.DTV   "5,5,5,5";   # Dwell-Time vector
DET.PAT2.DTM   "1,1,1,1";   # Dwell-Time modification flags

DET.PAT3.NAME  "Read";
DET.PAT3.NSTAT 4;
DET.PAT3.CLK1  "0000";
DET.PAT3.CLK2  "1001";
DET.PAT3.CLK3  "0110";      # Convert
DET.PAT3.CLK4  "0000";      # Start pulse
DET.PAT3.CLK5  "0000";
DET.PAT3.DTV   "5,5,5,5";   # Dwell-Time vector
DET.PAT3.DTM   "1,1,1,1";   # Dwell-Time modification flags

# up to ngcdcsSEQ_MAX_PAT (=2048) clock patterns in this format…
```

In order to keep the pattern definitions more compact the ***DET.CLK.MAPi*** keywords can be used to map the logical clocks defined in each pattern block onto physical clock lines. The mechanism is:

"*Physical clock line for logical clock* ***n***" = MAP[***n***].

The default clock state for all undefined clocks is "*low*" (0). Not mapped clocks can still be defined with their physical clock line number. E.g. when "*clock 63*" was needed only in "*pattern 3*" it could just be added there with

```
DET.PAT3.CLK63  "0001";
```

So the usage of the ***DET.CLK.MAPi*** keywords is not mandatory but simplifies editing and re-mapping.

In the "*.clk*" file the initial state for each clock is given with the ***DET.PAT.INITi*** keyword.

```
DET.PAT.INIT1  0;
DET.PAT.INIT2  1;
DET.PAT.INIT3  0;
DET.PAT.INIT4  0;
DET.PAT.INIT5  0;
:
up to 64 clock initialization states
```

The index refers to the logical clock number (as mapped with the ***DET.CLK.MAPi*** keywords). "***DET.PAT.INIT1 0***" means that clock 1 is "0", "***DET.PAT.INIT2 1***" means that clock 2 is "1" in the initialization pattern. All clocks not specified are zero. So one only needs to enter the ones which should be high after reset.

16.2. Sequencer Program

The sequencer program defines the order of execution of the defined clock patterns. The sequencer program may depend on an application specific set of parameters (like detector integration time, number of samples, window parameters…), which at runtime are only known to the detector control server. In the simplest case these parameters directly fit into the repetition counters of the ***LOOP*** and ***EXEC*** instructions:

```
LOOP $DET.NREADS
      EXEC <1_second_delay_pattern> $DET.EXPTIME
      EXEC <readout_pattern> 1
END
```

Other applications may need to compute the repetition counters via arithmetic formulas:

```
N = $DET.NREADS * $DET.NCYCLES;

LOOP $N
      EXEC <1_second_delay_pattern> $DET.EXPTIME
      EXEC <readout_pattern> 1
END
```

The detector readout will typically not consist of a single pattern but of another piece of sequencer program code which is assembled in a sub-routine:

```
N = $DET.NREADS * $DET.NCYCLES;
NX1 = $DET.NX / $DET.NCHANNELS

LOOP $N
      JSR <readout>
      EXEC <1_second_delay_pattern_1s> $DET.EXPTIME
      JSR <readout>
END
RETURN

<readout>:
LOOP $DET.NY
      EXEC <line_start_pattern> 1
      EXEC <sample_pattern> $NX1
END
RETURN
```

The next level of complexity is reached, when the computed parameters depend on the execution time of such a sub-routine:

```
N = $DET.NREADS * $DET.NCYCLES;
NX1 = $DET.NX / $DET.NCHANNELS;
T_DELAY = (($DET.EXPTIME / $N) - $time_r(<readout>)) * 1.0e8

LOOP $N
      JSR <readout>
      EXEC <10_nanosecond_delay_pattern> $T_DELAY
      JSR <readout>
END
RETURN

<readout>:
LOOP $DET.NY
      EXEC <line_start_pattern> 1
      EXEC <sample_pattern> $NX1
END
RETURN
```

The arithmetic formulas may use conditional instructions to compute results depending on the setting of logical parameters. It is also required that some of the computed parameters are passed back to the server. For example the "*actual*" exposure time or the "*minimum*" detector integration time can only be determined within this programming context, as arbitrary constant delays may be added such as chopper transition time or fixed delays after detector reset ("*DITDELAY*").

The evaluation of arithmetic formulas at run-time is implemented via the TCL scripting language. TCL has been chosen because this is a commonly used VLT software standard. Also the startup overhead is very short. The script evaluation may not be required by all applications. Where no script is required a simple parsing can be done. To optimize for both needs, a hybrid sequencer program format is used. The script evaluation between the ***SCRIPT***/***SCRIPT_END*** instructions can simply be skipped (as in the *test1.seq* example).

So finally the sequencer program consists of three parts: the **declaration-section**, the **evaluation-section** and the actual **program-code**.

The **declaration-section** contains the assignment of pattern names to pattern numbers, the declaration of the used parameters and the declaration of subroutines. Parameters needed in the **evaluation-section** have to be declared with the ***USE*** statement:

```
USE DET.PARAM1 DET.PARAM2 …
```

Parameters which are not used in the script but which are directly referred in the program code need not to be declared.

In case the execution time of such a subroutine is needed in the evaluation section the subroutines have to be declared with the ***SUBRT*** statement:

```
SUBRT routine1 routine2 …
```

Both the ***USE*** and the ***SUBRT*** statement can be repeated.

The **evaluation-section** is enclosed with ***SCRIPT***/ ***SCRIPT_END*** statements. The whole section can be skipped if it is not required. Control server parameters are passed to the script through the **svar(***parameter-name***)** array variable. The parameter values can be changed in the script and are passed back to the control server. Additional local parameters may be added to the **svar()** array and can be used within the **program-code** with $*parameter-name*. The local parameters are not passed back to the control server. Parameters in the ***DET.SEQi*** category are always used without the index as the index is stripped before the sequencer program is loaded into a certain sequencer instance. Setup parameters which are not within the scope of the control server but which are declared in the dictionary can still be accessed. If no value has been assigned yet via a ***SETUP*** command the program will return an error. This can be caught by assigning a default value with the following code:

```
if {![info exists svar(DET.MYVAR)]} {
    set svar(DET.MYVAR) 1
}
```

The execution times of the declared subroutines are stored in the **$time_r(***routine-name***)** array variable. The execution times of the declared patterns are stored in the **$time_p(***pattern-name***)** array variable. The unit is milliseconds. It has to be considered that all (local and global) floating-point parameters which are used afterwards in the **program-code** are rounded towards the nearest integer value before being applied as repetition factors or loop counters.

The **evaluation-section** is directly followed by the actual **program-code**. The loop repetition factors can be either a hard-coded decimal value, the value of a parameter or ***INFINITE***. A loop repetition factor evaluated to -1 is the same as ***INFINITE***, other negative values are illegal. The ***EXEC*** instruction is followed by two arguments: the pattern-name/-number and also a repetition factor. Both can be hard-coded values or the values of a parameter. A repetition factor of 1 may be omitted. The pattern number can be replaced by its name when a name was assigned in the declaration section. Similarly the ***JSR*** instruction is also followed by two arguments: the routine name and the repetition factor. Both can be hard-coded values or the values of a parameter. Again a repetition factor of 1 may be omitted.

Subroutines have to be labeled by a routine name. The label is just the routine name followed by ‘:’. The routine name must not contain special characters such as “white space”, “tab”, “#”, “$”. When used in the evaluation section the name must follow the TCL-rules for array variable names. The routine consists of normal program-code terminated via a ***RETURN*** statement. It is possible to include another sequencer program at any position using the ***INCLUDE*** <*file-name*> instruction. Typically this is used for sub-routine code (e.g. “*reset.seq*”, “*read.seq*”) which is then included in several main programs.

Declaration Section:

```
<pattern_name 1> = <pattern_number 1>
<pattern_name 2> = <pattern_number 2>
…
<pattern_name N> = <pattern_number N>

USE <parameter_name 1> <parameter_name 2> … <parameter_name N>
USE …

SUBRT <routine_name 1> <routine_name 2> … <routine_name N>
SUBRT …
```

Evaluation Section:

```
SCRIPT
[SETLIB {myTclLib1 myTclLib2 …}] # optional, can be used to embed own tcl-libraries
…
[Script Code]
# parameters are in $svar(parameter name)
# subroutine execution times (milliseconds) are in $time_r(routine_name)
# pattern execution times (milliseconds) are in $time_p(pattern_name)
…
SCRIPT_END
```

Remark: *It is possible to set individual clock-state vectors from within the script code using the svar(DET.PATi.CLKi) keywords.*

Program Code:

```
# Nested Loops
LOOP <INFINITE|repetition_factor|$parameter_name> [LOOP… …END]  END

# Repeated Pattern Execution
EXEC <pattern_name|pattern_number|$parameter_name> <repetition_factor|$parameter_name>

# Jump to Subroutine
JSR <routine_name> <repetition factor>

# Include another program file to this position
INCLUDE <file_name>

# Subroutine Label (Routine Name)
<routine_name>:

# Return from Main Program or from Subroutine
RETURN
```

Caution:
The limits for the loop counters and repetition factors are:

LOOP: 65535 ($2^{16} - 1$)
EXEC: 2147483647 ($2^{31} - 1$)
JSR: 65535 ($2^{16} - 1$)

The maximum number of nested loops (nesting depth) is 128.

Sequencer Program Example:

```
PAT_A = 1
PAT_B = 2
PAT_C = 3

USE PARAM1 PARAM2
USE PARAM3 PARAM4

SUBRT routine1 routine2

SCRIPT
if {$svar(PARAM4)} {
set svar(new1) [expr{$svar(PARAM1)*
               ($time_r(routine1) +
                $time_r(routine2))}]
} else {
set svar(new1) [expr{$svar(PARAM1)*
                $time_p(PAT_A)}]
}
SCRIPT_END

EXEC PAT_A $PARAM1
LOOP INFINITE
  JSR routine1 3
  LOOP $PARAM2
    LOOP $new1
      EXEC PAT_B
    END
  END
  JSR routine2
END
EXEC PAT_A 1
RETURN

routine1:
LOOP $PARAM3
  EXEC PAT_C 4
END
RETURN

routine2:
INCLUDE "test1.seq"
```

```
test1.seq:

PAT_D = 4
PAT_E = 5
PAT_F = 6

EXEC PAT_D
LOOP 10
  JSR myRoutine1
  LOOP 5
    EXEC PAT_E 4
  END
END
RETURN

myRoutine1:
LOOP 3
  EXEC PAT_F 10
END
RETURN
```

16.3. Break Points

The sequencer program can be stopped at defined points (e.g. after read-out) by inserting break-points. The break-points can be inserted at any place in any clock

pattern executed by the sequencer program by setting the "*break-point*" bit (***DET.PATi.CLK62***).

Example (four-state-pattern with break-point at the end):

```
DET.PAT4.NAME  "FrameSync";
DET.PAT4.NSTAT 4;
DET.PAT4.CLK1  "1100";
DET.PAT4.CLK2  "1010";
DET.PAT4.CLK3  "0110";
     :
DET.PAT4.CLK62 "0001";       # Break-Point
DET.PAT4.DTV   "5,5,5,5";    # Dwell-Time vector
DET.PAT4.DTM   "1,1,1,1";    # Dwell-Time modification flags
```

The break-points are activated whenever the *break* command-bit is set to one in the sequencer control register (see [RD1] for a detailed description). The sequencer will then stop at the next break-point. This is transparent to the user and can be enabled with the ***DET.SEQ.BREAK*** **<*T*|*F*>** keyword. When ***DET.SEQ.BREAK*** is set to '***F***' (default) then the sequencer will stop immediately when a "***SEQ –stop***" command is sent. When ***DET.SEQ.BREAK*** is set to '***T***' then the sequencer will stop at the next break-point when the "***SEQ –stop***" command is sent. The usage of break-points imposes the proper setting of a timeout. The timeout is specified in seconds with the ***DET.SEQ.BREAKTMO*** keyword. The default value is 10 seconds. A negative value indicates an infinite timeout (not recommended). A zero value lets the system choose. The zero value may cause additional overheads in order to compute the maximum time between two break-points. Also the system may then still switch to "*immediate stop*" when no proper timeout is found (e.g. no break-point within an infinite loop).

16.4. Voltage Setup

16.4.1. Voltage Configuration File

The voltage setup is done through a configuration file in short FITS format:

```
###############################################################################
# E.S.O. - VLT project
# "@(#) $Id$"
#
# who       when        what
# --------  ----------  ------------------------------------------
# jstegmei  2005-10-05  created
#
###############################################################################
#
# DESCRIPTION
# NGC voltage file. The keywords have to be defined for all
# used clock- and DC-voltages in the following format:
#
#     DET.CLDC.CLKHINMi   - Name of high clock i (optional)
#     DET.CLDC.CLKHIi     - Level of high clock i (mandatory)
#     DET.CLDC.CLKHIGNi   - Gain factor (optional)
#     DET.CLDC.CLKHIRAi   - Range of low clock i (mandatory)
#
#     DET.CLDC.CLKLONMi   - Name of low clock i (optional)
#     DET.CLDC.CLKLOi     - Level of low clock i (mandatory)
#     DET.CLDC.CLKLOGNi   - Gain factor (optional)
#     DET.CLDC.CLKLORAi   - Range of low clock i (mandatory)
#
#     DET.CLDC.DCNMi      - Name of DC voltage i (optional)
#     DET.CLDC.DCi        - Level of DC voltage i (mandatory)
#     DET.CLDC.DCGNi      - Gain factor (optional)
#     DET.CLDC.DCRAi      - Range for DC voltage i (mandatory)
#
###############################################################################

DET.CLDC.ILCK   F;      # External interlock mechanism

# Offsets:
DET.CLDC.CLKOFF 10.0;   # Global clock voltage offset (set to NAN for auto-
configuration)
DET.CLDC.DCOFF  10.0;   # Global DC voltage offset (set to NAN for auto-configuration)

# Clock Voltages:
DET.CLDC.CLKHINM1   "clk1Hi";
DET.CLDC.CLKHI1      3.000;
DET.CLDC.CLKHIGN1    1.0;
DET.CLDC.CLKHIRA1   "[-9.000, 9.000]";

DET.CLDC.CLKLONM1   "clk1Lo";
DET.CLDC.CLKLO1      0.000;
DET.CLDC.CLKLOGN1    1.0;
DET.CLDC.CLKLORA1   "[-9.000, 9.000]";

# up to 18 clock voltages like this …

# DC Voltages:
DET.CLDC.DCNM1   "DC1";
DET.CLDC.DC1      0.000;
DET.CLDC.DCGN1    1.0;
DET.CLDC.DCRA1   "[-9.000, 9.000]";

# up to 28 DC-voltages like this …
```

While the name and the range can only be entered through the voltage definition file itself, the actual voltage value may be changed within the given range at run-time via

SETUP command. The current voltage setup can be saved again to a file in this format ("***CLDC –save <filename>***" command, see section 4).When set to ***NAN***, the offset voltages ***DET.CLDC.CLKOFF*** and ***DET.CLDC.DCOFF*** will be set automatically by the server according to the given maximum ranges.

Instead of the ***DET.CLDC.CLKLOi***, ***DET.CLDC.CLKHIi***, ***DET.CLDC.DCi*** keywords (which define the values in Volts) the ***DET.CLDC.CLKLOXi***, ***DET.CLDC.CLKHIXi*** and ***DET.CLDC.DCXi*** keywords can be used in order to set the voltage register values directly. In that case the value is interpreted as hexadecimal value.

It is also possible to specify the individual DAC and telemetry channels for each clock and each bias. The keywords are ***DET.CLDCi.CLKLODACi***, ***DET.CLDCi.CLKHIDACi***, ***DET.CLDCi.DCDACi*** for the DAC channels and ***DET.CLDCi.CLKLOTELi***, ***DET.CLDC.CLKHITELi***, ***DET.CLDCi.DCTELi*** for the telemetry channels. Setting the channel number to a negative value (e.g. "-1") explicitly marks the DAC or the telemetry as "*not used*".

It is possible to define an initial voltage configuration (***DET.CLDCi.INITFILE***) which is applied whenever the voltage output goes from *disabled* state to *enabled* state before loading the final voltage configuration (***DET.CLDCi.FILE***).

16.4.2. Voltage Gain

The keywords

```
DET.CLDCi.CLKLOGNi
DET.CLDCi.CLKHIGNi
DET.CLDCi.CLKGNi
DET.CLDCi.DCGNi
```

define the output voltage gain for the clocks and the biases:

$$V_{dac} = V_{set} / gainFactor$$

V_{dac} is the voltage programmed in the DAC and V_{set} is the value entered by the user. When the last index is missing the gain is applied to all clock/bias voltages. The gain can be set individually for each voltage channel in the voltage configuration file. The keywords ***DET.CLDCi.CLKLOGNi*** and ***DET.CLDCi.CLKHIGNi*** refer to the low/high level of a clock. ***DET.CLDCi.CLKGNi*** refers to both levels.

16.4.3. Telemetry Gain

The keywords ***DET.CLDCi.TELCLKGN*** and ***DET.CLDCi.TELDCGN*** specify the telemetry gain for the clocks and the biases:

$$V_{tel} = V_{adc} * gainFactor$$

V_{tel} is the value shown to the user and V_{adc} is the value measured with the telemetry ADC. The gain is applied to all clock/bias channels.

16.4.4. Voltage Telemetry

The voltage telemetry is done

- once when going ***ONLINE*** and ***DET.CLDCi.MARGIN*** > 0.0
- when explicitly requested ("Telemetry"-button, "tel" command)
- whenever an exposure is started ***and*** after the last telemetry
  - any voltage has changed, or
  - the global DC-offset or the global CLOCK-offset has changed, or
  - the voltage output had been enabled or disabled, or
  - the system had been reset

The telemetry can be disabled by setting the telemetry channel to -1 in the voltage configuration file:

```
DET.CLDCi.CLKLOTELi -1; # clock low level telemetry channel
DET.CLDCi.CLKHITELi -1; # clock high level telemetry channel
DET.CLDCi.DCHITELi  -1; # bias voltage telemetry channel
```

This must be done for all channels for which no telemetry is available. Otherwise the automatic voltage check may fail and the system cannot go to ***ONLINE*** state.

The telemetry is checked against the voltage setup automatically when going to ***ONLINE*** state or when a new voltage configuration is loaded. The ***DET.CLDCi.MARGIN*** keyword defines the maximum value (in Volts) the telemetry may deviate from the setup. A margin of 0.0 disables the check. The voltage check may take some time until the DAC output has settled. The ***DET.CLDCi.DACTIME*** keyword defines a time interval (in seconds) since the last voltage setup until the telemetry must be in range. Otherwise the system does not go to ***ONLINE*** state (or goes to sub-state "***error***" when it was already online). The ***DET.CLDCi.MARGIN*** and ***DET.CLDCi.DACTIME*** keywords shall be set in the system configuration file (see section 17.1.2).

The margin may also be set individually for each clock- or bias-voltage using the keywords:

```
DET.CLDC.CLKLOi.MARGIN
DET.CLDC.CLKHIi.MARGIN
DET.CLDC.DCi.MARGIN
```

Setting the margin to 0.0 will again disable the voltage check for the given channel. The parameters may be added to the voltage configuration file.

16.4.5. Voltage Calibration

The control server can perform an automatic voltage calibration procedure to compensate channel specific offsets. The procedure is called with the command:

```
CLDC –module <n> -calibrate
```

The calibration is cleared with the command:

```
CLDC –module <n> -clear
```

The calibration procedure computes the differences between setup value and voltage telemetry and then subtracts the computed values from the values of the setup parameters (***DET.CLDC.CLKLOi/CLKHIi/DCi***).

Instead of this automatic procedure the offset compensation can also be set manually with the parameters:

```
DET.CLDC.CLKLOERRi
DET.CLDC.CLKHIERRi
DET.CLDC.DCERRi
```

There is a hard limit of {`+-200mv * gain`} for the compensation value in order not to give the possibility to compensate major malfunctions or to go far beyond the defined ranges. The error compensation parameters can also be specified in the voltage configuration file which in this case will become board-specific. When the automatic calibration is commanded with "***CLDC –module <n> –calibrate***" then the value from the voltage configuration file will be overwritten.

The error compensation keywords are fully supported by the voltage configuration file handler – i.e. once they are set (via configuration file, setup parameter or automatically by the calibration function) they are written to a file like any other voltage keyword when the actual configuration is saved with the command:

```
CLDC –module <n> -save <filename>
```

Additionally the error vectors can be saved to a file with the command:

```
CLDC –module <n> -sverr <filename>
```

This generates a setup-file which can be loaded again with the command "***SETUP –file <filename>***".

There is also a keyword to choose automatic voltage calibration whenever a new voltage file is loaded (***DET.CLDCi.AUTOCAL T|F***). Due to the long DAC-settling time this is time consuming and the preferred way is to do the calibration once per board and to save the values for later usage.

16.4.6. Overvoltage Protection

The detector is protected against overvoltage by a protection circuit. The circuit disables the voltage output in case a clock- or bias-voltage exceeds a given absolute range. When status polling is switched on (***DET.CON.POLL*** = ***'T'***) then the server periodically reads the status register and will go to error-state in case the protection circuit has switched off the voltage output.

16.4.7. External Interlock

The front-end basic board provides an interlock mechanism which allows the voltage output to be disabled via an external signal (e.g. from a temperature controller). The interlock mechanism can be enabled or disabled via a configuration keyword (***DET.CLDCi.ILCK T|F***). The keyword is valid both in the system- and in the detector configuration file. Furthermore it can explicitly be set in the voltage configuration file in order to declare a certain voltage configuration as "*safe*" (***DET.CLDC.ILCK = 'F'***) or as "*critical*" (***DET.CLDC.ILCK = 'T'***). A "*safe*" configuration will disable the interlock after being uploaded to the controller. A "*critical*" configuration will enable the interlock before being uploaded to the controller. The interlock cannot be controlled via ***SETUP***-command. By default the interlock is disabled. When status polling is switched on (***DET.CON.POLL*** = ***'T'***) then the server periodically reads the status register and will go to error-state in case the interlock has switched off the voltage output.

Caution:
The interlock handling is implemented only in boards with firmware revision 10.1 or later. It is not possible to load a configuration with the interlock being enabled into boards with earlier firmware revision ("voltage setup rejected").

16.5. Read-Out Modes

The infrared detectors can be read out in various ways. The read-out scheme (multiple sampling, non-destructive, etc.) is given by the sequencer programs loaded into each sequencer instance. The acquisition process has to apply the appropriate processing algorithm for each sequencer program. There may be multiple acquisition processes associated to one sequencer (i.e. each one is processing a part of the frame/mosaic). Such a combination of read-out program and processing is called "*read-out mode*". The read-out modes are defined in the detector configuration file (see section 17.2) 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 process(es) 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. When the system is in ***ONLINE*** state the read-out mode can be selected via ***SETUP*** command either by name (***DET.READ.CURNAME***) or by its assigned id (***DET.READ.CURID***).

Caution:
The read-out mode name should never contain white spaces as this may cause messy ***SETUP*** *commands when using the* ***DET.READ.CURNAME*** *keyword. White spaces in the read-out mode name are automatically replaced with underscore characters '_' when the detector configuration file is loaded.*

Changes with respect to IRACE:

The keywords for defining and for selecting the read-out modes have changed:

DET.NCORRS	->	*DET.READ.CURID*
DET.NCORRS.NAME	->	*DET.READ.CURNAME*
DET.IRACE.RMDEF	->	*DET.READ.DEFAULT*
DET.IRACE.RMi.<XXX>	->	*DET.READi.<XXX>*

17. Configuration

Templates for configuration files (including sequencer programs and voltage configuration files as described in the previous sections) are provided in the ***ngcdcs*** software module:

```
ngcdcs/templates/xxdcfg
```

The ***xxdcfg*** template is installed in:

```
$INTROOT/templates/forNGC/
$VLTROOT/templates/forNGC/
```

Both system- and detector-configuration files allow the usage of *include-files*. The files can be inserted at any point with the ***DET.INCLUDE*** keyword. The path is either absolute or relative to the parent file. The nesting depth is limited to 5 layers.

17.1. Controller Electronics System Configuration

The controller electronics 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 system configuration includes all information to identify the hardware configuration including the interface device names and the computing architecture (host names, environments, etc.). In addition it also defines the default detector configuration file, the default values for file format and naming scheme and some keywords describing the general system behavior like starting the sequencer(s) automatically when going to ***ONLINE*** state (***DET.AUTOSTRT*** = "***T***").

Before going to ***ONLINE*** state the actual controller electronics is checked against this system configuration and an error is reported in case something does not match.

17.1.1. Device Configuration

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" defined by the ***DET.DEVi.SRV*** keyword).

Finally an optional device type (***DET.DEVi.TYPE***) can be given when a communication interface other than the CCS message system should be used to control the driver interface process. This has to bet set to "*socket*" in case no CCS environment is running on the NGC-LLCU:

```
DET.DEV1.NAME    "/dev/ngc0_com"; # associated device name
DET.DEV1.HOST    "<host-name>";   # host where interface resides
DET.DEV1.ENV     "";              # environment name
DET.DEV1.SRV     "";              # server name (optional)
DET.DEV1.TYPE    "socket";        # type
```

When a CCS environment is running on the NGC-LLCU one should use:

```
DET.DEV1.NAME    "/dev/ngc0_com"; # associated device name
DET.DEV1.HOST    "<host-name>";   # host where interface resides
DET.DEV1.ENV     "<env-name>";    # environment name
DET.DEV1.SRV     "";              # server name (optional)
DET.DEV1.TYPE    "";              # type
```

17.1.2. Module Configuration

Each of the controller electronics modules (***SEQ***, ***CLDC***, ***ADC***) gets one interface device assigned, through which it is accessed. The assignment is done via a reference index defined by the ***DET.SEQi.DEVIDX***, ***DET.CLDCi.DEVIDX***, ***DET.ADCi.DEVIDX*** keywords and a linking route *through* the interface *to* the target module. For the moment only chaining routes (*Header-5* = "*Down-Link 1*") are supported. So the route defines *n*-times "*5*," for *n* modules routed through. It is always terminated by "*2*" (*Header-2* = "*this board*"). For each of the modules an optional name can be defined. So finally the server will just see linear lists of sequencer-, CLDC- and ADC-modules independent from the nesting structure of the NGC controller electronics module network(s).

Example for CLDC-module:

```
DET.CLDC1.DEVIDX     1;        # associated device index
DET.CLDC1.ROUTE      "5,2";    # route to module (2nd board in chain)
DET.CLDC1.AUTOENA    F;        # auto-enable at online
DET.CLDC1.MARGIN     0.5;      # margin for voltage check (in Volts)
DET.CLDC1.DACTIME   10.0;      # max. DAC settling time (in seconds)
DET.CLDC1.DCGN       2.0;      # bias gain
DET.CLDC1.CLKGN      1.0;      # clock gain
DET.CLDC1.TELDCGN    2.0;      # telemetry gain (biases)
DET.CLDC1.TELCLKGN   1.0;      # telemetry gain (clocks)
DET.CLDC1.ILCK       T;        # external interlock
DET.CLDC1.NAME      "CLDC-1"; # module name (optional)
```

By default the voltages on the CLDC modules are not enabled when going. This behavior can be changed by setting ***DET.CLDCi.AUTOENA*** to "***T***" or "***F***".

The ***DET.CLDCi.MARGIN*** keyword defines the margin (in Volts) for the automatic telemetry check when going to ***ONLINE*** state or when a new voltage configuration is loaded (see section 16.4).

Example for two sequencer modules:

```
DET.SEQ1.DEVIDX     1;              # associated device index
DET.SEQ1.ROUTE      "2";            # route to module
DET.SEQ1.NAME       "Sequencer 1";  # module name (optional)
DET.SEQ1.GROUP      "SEQ";          # group name (optional)
DET.SEQ2.DEVIDX     1;              # associated device index
DET.SEQ2.ROUTE      "5,2";          # route to module
DET.SEQ2.NAME       "Sequencer 2";  # module name (optional)
DET.SEQ2.GROUP      "SEQ";          # group name (optional)
```

The sequencer modules can be grouped by specifying a group name with the ***DET.SEQi.GROUP*** keyword. All sequencers with the same group name will forward all setup parameters to each other to have their configuration aligned at all times. It is recommended to use the grouping when the sequencers are running synchronously, i.e. ***DET.SEQi.RUNCTRL***='***T***'. The grouping does not affect the ***DET.SEQi.RUNCTRL*** and ***DET.SEQi.CVTEXT*** keywords, which are by definition always specific for each instance.

Example for ADC module:

```
DET.ADC1.DEVIDX     1;        # associated device index
DET.ADC1.ROUTE      "2";      # route to module
DET.ADC1.NAME       "ADC 1";  # module name (optional)
DET.ADC1.NUM        4;        # total number of ADC units on board
DET.ADC1.BITPIX    16;        # number of bits per pixel
DET.ADC1.FIRST      T;        # first in chain
DET.ADC1.PKTCNT     0;        # packet routing length
DET.ADC1.CLAMP      F;        # clamp & sample mode
DET.ADC1.FILTER     0;        # filter (0 or 1)
```

The ***DET.ADCi.PKTCNT*** keyword defines the number of packets the ADC module has to wait for until it can send out its own data. The ***DET.ADCi.FIRST*** keyword must be set to "***T***" for the first ADC module in the chain. The filter can only be switched when not operating in "*clamp & sample*" mode.

It is possible to prevent individual modules from exporting their status to the FITS header. The following keywords can be set in the system configuration file:

```
DET.SEQi.FITS   T|F; # FITS enabled
DET.CLDCi.FITS  T|F; # FITS enabled
DET.ADCi.FITS   T|F; # FITS enabled
```

When set to "***F***" then the status of the respective module is not exported to the FITS header. The default value of these keywords is "***T***".

17.1.3. Acquisition Modules

The acquisition modules also have to be declared in the system configuration file. This does not yet define the actual process that will be launched by the module. But it defines the host (***DET.ACQi.HOST***) where the process will be launched. This is not necessarily the same as the host for the controller device (***DET.DEVi.HOST***) – but for simple system this is normally the case.

The modules are attached to individual sequencer instances with the ***DET.ACQi.SEQIDX*** keyword. Each module should refer 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. A zero or negative index tells the system not to associate the acquisition module to any sequencer.

It is possible to read the data from multiple DMA devices into one acquisition module. This is indicated by the device index of the ***DET.ACQi.DEVi*** keyword.

```
DET.ACQ1.DEV1      "/dev/ngc0_dma"; # DMA device 1 name
DET.ACQ1.DEV2      "/dev/ngc1_dma"; # DMA device 2 name
DET.ACQ1.HOST      "$NGCPP_HOST";   # host name for acq.-process
DET.ACQ1.CMDPORT   0;               # acq.-process command port
DET.ACQ1.DATAPORT  $NGCPP_DATA;     # acq.-process data port
DET.ACQ1.NCLIENT   2;               # max. data server clients
DET.ACQ1.SEQIDX    1;               # associated sequencer instance
```

17.1.4. Reference Configuration

To fully verify a given system a reference configuration can be added to the system configuration file. This defines serial numbers, type, version and revision for each board in the system including the PCI/PCIe interface device. When going to ***ONLINE*** state the SW will verify that all given reference keywords match the actual system. Valid keywords are:

```
DET.NUMDEV              - number of PCI/PCIe interface devices
DET.DEVi.SERNO          - serial number of the device
DET.DEVi.REV            - revision of the device
DET.DEVi.NUMBOARD       - number of boards connected to device
DET.DEVi.BOARDi.TYPE    - board type
DET.DEVi.BOARDi.VERSION - board version number
DET.DEVi.BOARDi.REV     - board revision
DET.DEVi.BOARDi.SERNO   - serial number of board
DET.DEVi.BOARDi.BACK    - serial number of backplane
DET.DEVi.BOARDi.TRANS   - serial number of transition board
```

The board type may be one of "*FEB*", "*AQ32*", "*HIGH-SPEED*", "*AOWFS*" or a one digit hexadecimal number for any undefined type. The version number defines the variant of the board type (e.g. CCD or IR setup).

Example:

```
DET.NUMDEV              1;
DET.DEV1.NUMBOARD       2;
DET.DEV1.SERNO          "7FF301A11415";
DET.DEV1.REV            "2.1.1";
DET.DEV1.BOARD1.TYPE    "FEB";
DET.DEV1.BOARD1.VERSION 0;
DET.DEV1.BOARD1.REV     "5.5.2";
DET.DEV1.BOARD1.SERNO   "87F301021B10";
DET.DEV1.BOARD1.BACK    "3F0401B12412";
DET.DEV1.BOARD1.TRANS   "0FA802A173C4";
DET.DEV1.BOARD2.TYPE    "AQ32";
DET.DEV1.BOARD2.VERSION 0;
DET.DEV1.BOARD2.REV     "4.1.0";
DET.DEV1.BOARD2.SERNO   "170311A21211";
DET.DEV1.BOARD2.BACK    "3F0401B12412";
DET.DEV1.BOARD2.TRANS   "06A18AA172C2";
```

Caution:
Only those boards and board components are checked which have a keyword defined in the system configuration file.

17.1.5. Example File

```
# Server configuration
DET.DETCFG          "test.dcf";     # default detector configuration (optional)

# Exposure frame configuration (optional)
DET.FRAM.FORMAT     "extension";    # default FITS-file format
DET.FRAM.MULTFILE    F;             # generate multiple files
DET.FRAM.NAMING     "request";      # default FITS-file naming scheme
DET.FRAM.LRGFILE     F;             # large file extension

# Device description - other DEV category keywords can be added here
DET.DEV1.NAME    "/dev/ngc0_com";   # associated device name
DET.DEV1.HOST    "$NGCPP_HOST";     # host where interface resides
DET.DEV1.ENV     "";                # environment name
DET.DEV1.TYPE    "socket";          # type

# CLDC modules - other CLDC category keywords (e.g. from section 17.2) can be added
here
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.TELDCGN   3.0;            # telemetry gain (biases)
DET.CLDC1.TELCLKGN  1.0;            # telemetry gain (clocks)
DET.CLDC1.DCGN      2.0;            # bias gain
DET.CLDC1.CLKGN     1.0;            # clock gain
DET.CLDC1.NAME      "CLDC 1";       # module name (optional)

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.TELDCGN   3.0;            # telemetry gain (biases)
DET.CLDC2.TELCLKGN  1.0;            # telemetry gain (clocks)
DET.CLDC2.DCGN      2.0;            # bias gain
DET.CLDC2.CLKGN     1.0;            # clock gain
DET.CLDC2.NAME      "CLDC 2";       # module name (optional)

# Sequencers - other SEQ category keywords (e.g. from section 17.2) can be added here
DET.SEQ1.DEVIDX     1;              # associated device index
DET.SEQ1.ROUTE      "2";            # route to module
DET.SEQ1.NAME       "Sequencer 1";  # module name (optional)
DET.SEQ1.GROUP      "SEQ";          # group name (optional)

# ADC modules - other ADC category keywords (e.g. from section 17.2) can be added here
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 (# of packets from down-
link)
DET.ADC1.NAME      "ADC-Module 1";  # module name (optional)

DET.ADC2.DEVIDX     1;              # associated device index
DET.ADC2.ROUTE      "5,2";          # route to module
DET.ADC2.NUM       32;              # number of enabled ADC units on board
DET.ADC2.BITPIX    16;              # number of bits per pixel
DET.ADC2.FIRST      F;              # first in chain
DET.ADC2.PKTCNT     0;              # packet routing length (# of packets from down-
link)
DET.ADC2.NAME      "ADC-Module 2";  # module name (optional)
```

```
# Acquisition modules – other ACQ category keywords can be added here
DET.ACQ1.DEV        "/dev/ngc0_dma"; # DMA device name
DET.ACQ1.HOST       "$NGCPP_HOST";   # host name for acq.-process
DET.ACQ1.CMDPORT    0;               # acq.-process command port
DET.ACQ1.DATAPORT   0;               # acq.-process data port
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       "$NGCPP_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
```

17.2. Detector Configuration

The detector configuration describes the usage of the system with respect to the connected detector(s). It is given in a configuration file (short FITS format), which can be loaded at server run-time via a ***SETUP*** command (***DET.DETCFG*** <*filename*>). When the system is in ***ONLINE*** state the configuration is directly applied to the controller electronics. 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(s). The ***DET.CHIPS*** keyword defines the number of chips in a mosaic (default value is "1"). There is also a keyword to assign a 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 ***DET.ACQi.SPLITX/Y*** keywords define for each acquisition module into how many chips the produced image shall be split into x- and y-direction (default is one chip per image – i.e. both values are 1). See section 9.8 for details concerning the mosaic setup.

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 dwell time (***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. they will propagate and react on the external run-signal). The ***DET.SEQi.CVTEXT*** keyword defines whether the external convert line is enabled or disabled. 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) (see section 16.5).

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
DET.SEQ1.CVTEXT    T;             # external convert

# ADC module setup
DET.ADC1.DELAY      0;            # conversion strobe delay (ticks)
DET.ADC1.OFFSET     2.0;          # ADC offset (Volt)
DET.ADC1.ENABLE     4;            # number of enabled ADC units
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.ACQ1  "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";
```

17.3. Special Configurations

The control sever can be configured for pure HW-control (i.e. no data acquisition and exposure handling) or also for pure data acquisition (e.g. hardware is controlled by another instance).

17.3.1. HW-Control Sub-System

In order to let the control server do only the hardware control but no data acquisition the controller electronics system configuration does not define any acquisition module. There are no ***DET.ACQi*** category keywords in this case. The ***DET.FRAM*** category keywords will be ignored and can also be omitted.

With such a configuration the data acquisition and exposure handling is no more within the scope of the control server, but the hardware can still be operated without restriction as described in the previous sections. This makes it possible to run the control server as a command driven sub-system of any DCS using the NGC (e.g. the *NGCOSW*).

The "*exposure*", "*mode*", "*guiding*" and "*acq*" database branches (see section 6) exist, but are not in use when the system is configured for HW-control only.

17.3.2. Data Acquisition Sub-System

In order to let the control server do only the data acquisition but no hardware control, the device definitions (***DET.DEVi***) and the controller electronics module definitions (***DET.SEQi, DET.CLDCi, DET.ADCi, …***) can be omitted from the system configuration file.

17.4. Multiple Data-Links

17.4.1. Side-Link Configuration

When the bandwidth on the data-link is too slow to transfer the data of all ADC-modules in the system then the data has to be sent out directly via a second link ("*side-link*"). The side-link is a normal communication- and data-link available on each NGC board.

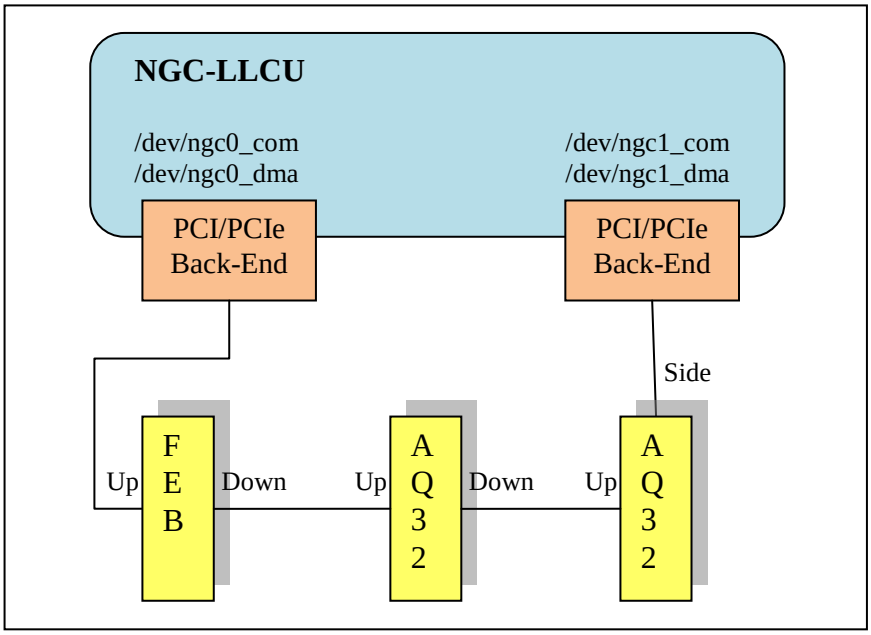

Figure 25 Side-Link Configuration

The ADC-module board must be configured to break-up the normal upstream-link and to open the side-link instead for further communication. The side-link then connects to a second PCI/PCIe back-end card (see Figure 25). To achieve this 3 additional device configuration keywords are provided:

Keyword	Type	Description
`DET.DEVi.SIDELINK`	`Logical`	When set to 'T' this instance is controlled via its side-link instead of its upstream-link. This requires a master and a configuration route to be specified.
`DET.DEVi.MASTER`	`Integer`	Index of the master device of this instance. Only applicable when the side-link is used.
`DET.DEVi.CFGROUTE`	`String`	Link route leading from the master device to this instance. Only applicable when the side-link is used.

Table 44 Side-Link Configuration Keywords

System Configuration Example (for Figure 25):

```
DET.DEV1.NAME      "/dev/ngc0_com";  # device name
DET.DEV2.NAME      "/dev/ngc1_com";  # device name
DET.DEV2.SIDELINK T;                 # switch to sidelink
DET.DEV2.MASTER   1;                 # index of master device
DET.DEV2.CFGROUTE "5,5,8";           # configuration route

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.FIRST     T;                # first in chain
DET.ADC1.PKTCNT    1;                # packet routing length
DET.ADC1.NAME      "ADC-FEB";        # module name (optional)

DET.ADC2.DEVIDX    1;                # associated device index
DET.ADC2.ROUTE     "5,2";            # route to module
DET.ADC2.NUM      32;                # number of enabled ADC units on board
DET.ADC2.FIRST     F;                # first in chain
DET.ADC2.PKTCNT    0;                # packet routing length
DET.ADC2.NAME      "AQ32-1";         # module name (optional)

DET.ADC3.DEVIDX    2;                # associated device index
DET.ADC3.ROUTE    "2";               # route to module
DET.ADC3.NUM      32;                # number of enabled ADC units on board
DET.ADC3.FIRST     T;                # first in chain
DET.ADC3.PKTCNT    0;                # packet routing length
DET.ADC3.NAME      "AQ32-2";         # module name (optional)

DET.ACQ1.DEV       "/dev/ngc0_dma"; # DMA device name
DET.ACQ2.DEV       "/dev/ngc1_dma"; # DMA device name
```

17.4.2. Multi-Channel Back-End

The PCI-express version of the back-end card support three additional communication channels which can be operated via the associated devices "*/dev/ngc<n>_comx1*", "*/dev/ngc<n>_comx2*" and "*/dev/ngc<n>_comx3*". The side-link configuration rules apply in the same way as described in section 17.4.1. The data-streams on these auxiliary channels can be multiplexed to one single "*/dev/ngc<n>_dma*" DMA-device thus avoiding the concurrent DMA-interrupt handling when acquiring data from multiple devices as described in section 17.1.3. The DMA channels in use are selected with the ***DET.ACQi.CHAN*** keyword (see Table 21). The 32-bits indicate up to 32 enabled data channels. If the keyword has a negative value (= *system default*) then the channel selection is disabled and the device driver default value will be used. The data order will always be:

<64-bits from 1st enabled channel>
<64-bits from 2nd enabled channel>
<…>
<64-bits from Nth enabled channel>

where "*N*" is the number of auxiliary DMA channels in use. An example with one auxiliary channel is shown in Figure 26.

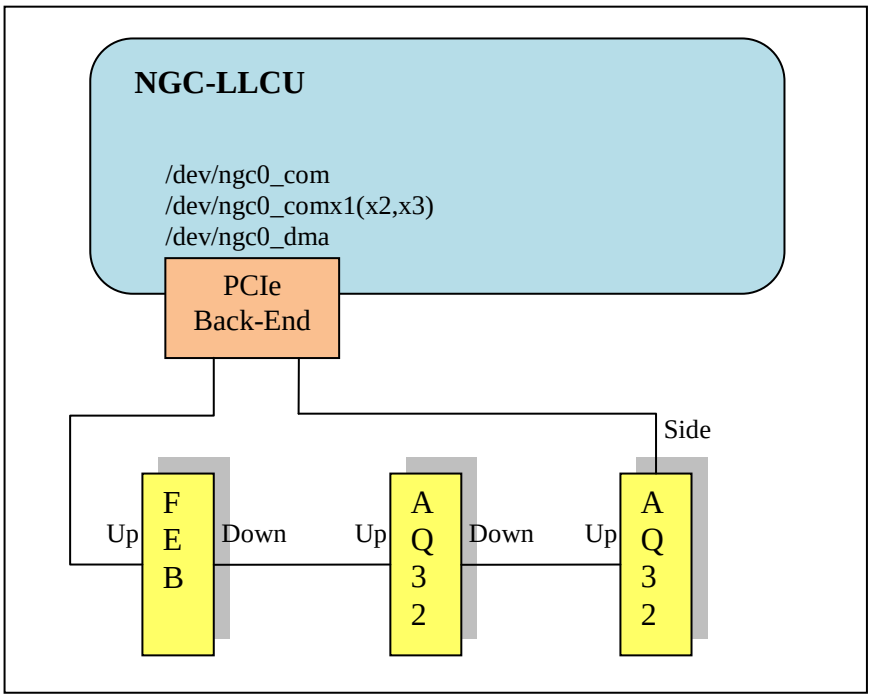

Figure 26 Multi-Channel Back-End

System Configuration Example (for Figure 26):

```
DET.DEV1.NAME      "/dev/ngc0_com";    # device name
DET.DEV2.NAME      "/dev/ngc0_comx1"; # device name
DET.DEV2.SIDELINK T;                  # switch to sidelink
DET.DEV2.MASTER   1;                  # index of master device
DET.DEV2.CFGROUTE "5,5,8";            # configuration route

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.FIRST     T;                 # first in chain
DET.ADC1.PKTCNT    1;                 # packet routing length
DET.ADC1.NAME      "ADC-FEB";         # module name (optional)

DET.ADC2.DEVIDX    1;                 # associated device index
DET.ADC2.ROUTE     "5,2";             # route to module
DET.ADC2.NUM      32;                 # number of enabled ADC units on board
DET.ADC2.FIRST     F;                 # first in chain
DET.ADC2.PKTCNT    0;                 # packet routing length
DET.ADC2.NAME      "AQ32-1";          # module name (optional)

DET.ADC3.DEVIDX    2;                 # associated device index
DET.ADC3.ROUTE    "2";                # route to module
DET.ADC3.NUM      32;                 # number of enabled ADC units on board
DET.ADC3.FIRST     T;                 # first in chain
DET.ADC3.PKTCNT    0;                 # packet routing length
DET.ADC3.NAME      "AQ32-2";          # module name (optional)

DET.ACQ1.DEV       "/dev/ngc0_dma";   # DMA device name
DET.ACQ1.CHAN      0x3;               # DMA channel selection
```

17.5. Configuration Template

A complete configuration example is provided as template in the ***ngcircon*** software module:

```
ngcircon/templates/xxircfg
```

The ***xxircfg*** template source is installed in:

```
$INTROOT/templates/forNGC/
$VLTROOT/templates/forNGC/
```

To install the configuration:

```
cd xxircfg/src
make install
```

Then add the following lines to the ***DATABASE.db*** file:

```
#define ngcirconINSTANCE ngcircon
#define ngcirconROOT :Appl_data:ircam1
#include "ngcircon.db"

#define ngcirconINSTANCE ngcircon_ircam2
#define ngcirconROOT :Appl_data:ircam2
#include "ngcircon.db"
```

The *ngcirconROOT* may define any other (existing) path instead of "***:Appl_data:ircam1***" or "***:Appl_data:ircam2***".

Now the system can be started with

```
ngcdcsStartServer IRCAM1/2 [-gui]
```

IRCAM1 defines a system with one single detector (no instance definition, database is "*<alias>ngcircon*". *IRCAM2* defines a more complex second instance with two detectors (instance is "*ircam2*", database is "*<alias>ngcircon_ircam2*"). *IRCAM1* and *IRCAM2* can run in parallel. To start just the GUI use

```
ngcdcsStartGui IRCAM1/2
```

18. The I²C Bus Interface

In order to communicate with other sub-devices (e.g. the pre-amplifier board) the NGC DFE implements a two wire I²C bus interface. The two I/O-lines are fed into a control port register (address 0xC000) on the front-end basic board. The *read*- and *write*-accesses to the bus are done by reading from or writing to the I/O-line bits on this port.

18.1. The I²C Interface Class

The bus protocol timing is implemented via a general I²C interface class (***ngcbI2C***). The actual communication is done via a state word that has to be sent or received via appropriate call-back functions. The clock-line (SCL) and the data-line (SDA) of the I²C two-wire bus can be placed on any bit in the state word. The bit positions are assigned in the constructor. It may happen that different bits are used for the *read*- or the *write*-accesses. Furthermore the acknowledge signal (ACK) may be read from a third (artificial) bit. By default this is the same as the data-line (SDA) – according to the I²C standard. It is also possible to ignore the acknowledge signal at all.

The I²C clock-stretching capability is controlled via a polling counter (N). The clock-stretching times out when the clock-line (SCL) did not go high after N read attempts. Setting this counter to zero disables the clock-stretching capability and the clock-line is not polled after toggling. Additionally (if the counter has a positive value) the clock-stretching can be enabled/disabled separately for both *read*- and *write*-operations.

18.2. Engineering Interface

The driver interface process takes care of the I²C bus protocol timing whenever the I²C control port register space (0xC000 to 0xC3FF) is addressed. In that case it creates a new instance of the ***ngcbI2C*** class with the call-back functions for the state-I/O properly set. The clock-line (SCL) is placed on bit 0, the data-line (SDA) is placed on bit 1 and the acknowledge-signal (ACK) is read from bit 2. A direction switch for the data-line (SDA) is placed on bit 3. When bit 3 is set SDA becomes SDA_IN. The clock-stretching capabilities are disabled by default for both *read*- and *write*-operations.

The slave address is given in the lower 10-bits of the standard NGC DFE address. So the low-level interface looks like:

"*wraddr <route> 0xC<YYY> <value1> <value2> … <valueN>*"
"*rdaddr <route> 0xC<YYY> <value1> <value2> … <valueN>*"

In this case "*YYY*" is the slave address of the device on the I²C bus (see examples below). The *route* value specifies the board on which the control port register resides. The interface supports both the standard 7-bit slave addressing (see also Table 45) and the I²C 10-bit addressing extension. An arbitrary number of 8-bit values may follow.

In case the slave device is a 16-bit I^{2}C bus expander (address range 0xC020 to 0xC027) the values are automatically interpreted as 16-bit integer numbers with *Port-1* in the MSB and *Port-0* in the LSB. This conversion is done to avoid cases where non-even numbers of bytes are requested for such a device.

Caution:
The following slave addresses are reserved according to the I^{2}C standard:

0000000	General call address.
0000001	CBUS address.
0000010	Reserved for new bus format.
0000011	Reserved for future extensions.
00001XX	Reserved for future extensions.
11111XX	Reserved for future extensions.
11110XX	10-bit slave address extension.

Table 45 I^{2}C Reserved Slave Addresses

The I^{2}C bus protocol timing can still be tuned using the upper 16-bit (D16-31) of the standard NGC DFE address:

D16-17	Install a sleeping timer after each state I/O. The timer values are: 0 (00): no sleep 1 (01): 1 millisecond sleep 2 (10): 10 milliseconds sleep 3 (11): 100 milliseconds sleep
D18	Reserved.
D19	Reserved.
D20	Enable clock-stretching for *read*-operations.
D21	Enable clock-stretching for *write*-operations.
D22	Ignore acknowledge bit.
D23-31	Reserved.

Table 46 I^{2}C Protocol Tuning

Examples:

Write one 16-bit value to the I^2C port of the first DFE module with clock-stretching enabled and a sleeping timer of 1 millisecond (remember that *0xC021* is within the address range of the 16 bit I^2C bus expander):

"*wraddr 0x2 0x31C021 0xABCD*"

Write three 8-bit values to I^2C port of first DFE module with clock-stretching disabled and no sleeping timer:

"*wraddr 0x2 0xC010 0xAB 0xCD 0xEF*"

Write one 8-bit value to I^2C port of first DFE module with clock-stretching disabled, no sleeping timer and the acknowledge-bit being ignored:

"*wraddr 0x2 0x40C010 0xAB*"

Write one 16-bit value to the I^2C port of the third DFE module with clock-stretching disabled and a sleeping timer of 10 milliseconds (remember that *0xC020* is within the address range of the 16 bit I^2C bus expander):

"*wraddr 0x5 0x5 0x2 0x2C020 0xABCD*"

19. Preamplifier Interface

The ***ngcdcsEvh*** control server implements an interface to the optical preamplifier via the I^2C-Bus (see section 18).

19.1. Module Creation

The instantiating of the interface in the controller electronics system configuration is done via the following keywords:

```
DETi.PREAMPi.DEVIDX  <device index>;    # associated device index
DETi.PREAMPi.ROUTE   "<route>";         # route to module
DETi.PREAMPi.NUMCHAN <n>;               # number of channels
DETi.PREAMPi.NAME    "<name>";          # module name
DETi.PREAMPi.REV     <floating point>; # revision
```

The number of channels can be set from minimum 1 to maximum 8. The default value is 4. When using more than 4 channels an extension board is required. The control server takes care of the proper addressing. The revision specifies the version of the preamplifier which is actually implemented. The default value is 3.0. The control server takes care of mapping the parameter values to the proper register-bits depending on the revision.

Example:

```
DET.PREAMP1.DEVIDX  1;      # associated device index
DET.PREAMP1.ROUTE   "2";    # route to module
DET.PREAMP1.NUMCHAN 4;      # number of channels in use
DET.PREAMP1.NAME    "PA-1"; # module name
DET.PREAMP1.REV     "3.0";  # revision
```

19.2. Database

The run-time parameters of the preamplifier module are stored in database. The attributes are given in Table 15.

19.3. Setup Parameters

The preamplifier setup keywords are defined in Table 22. A mirror of the setup keywords is kept within the control server and can be retrieved via the ***STATUS*** command at any time. A preset value can be given in the controller electronics system configuration. The values are then applied when going to ***ONLINE*** state. There are two special keywords which allow the fine-tuning of the reference voltage DACs (DAC command register) and the I^2C-bus configuration (defining the upper 16-bit of the I2C-port as given in Table 46):

```
DETi.PREAMPi.VREFCMD <4-bit hex value>;      # DAC command
DETi.PREAMPi.BUSCMD  <16-bit hex value>;     # I2C-bus
configuration
```

19.4. Control Widget

The preamplifier interface can be operated through the NGC engineering GUI. The control widget is located in the notebook area:

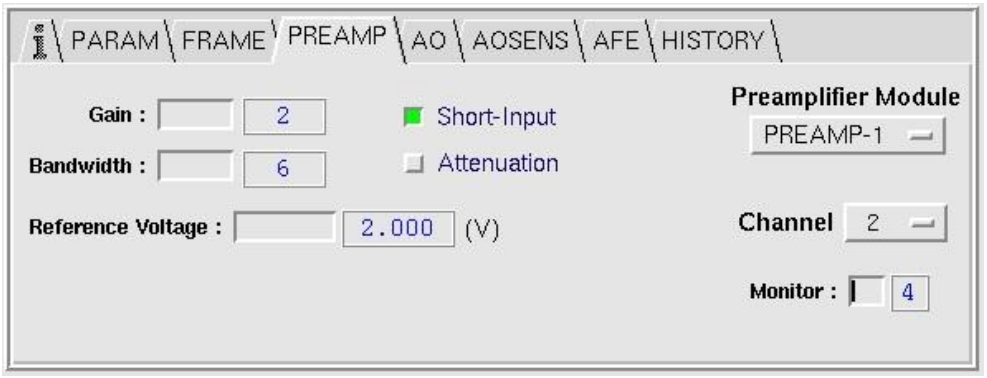

Figure 27 Preamplifier Control Widget

20. Shutter Interface

The ***ngcdcsEvh*** control server implements a low level shutter interface. The shutter(s) have to be declared in the system configuration file.

Example:

```
DET.SHUT1.DEVIDX    1;            # associated device index
DET.SHUT1.ROUTE     "2";          # route to module
DET.SHUT1.NAME      "Shutter-1"; # optional name
```

Then the registers of each shutter module can be accessed through ***SETUP/STATUS*** commands:

DET.SHUTi.REGCTRL - Shutter control register. The ***SETUP*** command accepts decimal, hexadecimal (0x…) or binary (0b…) values. The *STATUS* command returns a hexadecimal value of the registers software mirror (the pulsed bits are masked out).

DET.SHUTi.REGSTAT - Shutter status register. The ***STATUS*** command returns a hexadecimal value (0x…).

DET.SHUTi.EXPTIME - Exposure time register. A decimal value is used for both *SETUP* and *STATUS* command.

DET.SHUTi.EVTCNTi - Event counters (4 registers). The ***STATUS*** command returns a decimal value.

All registers are 32-bits wide. The register contents are not masked. The exposure time and the event counters are given in ticks. The current resolution is 10 microseconds per tick but may change in future applications. See [RD1] for a detailed description of the register contents and functionalities.

21. Guiding Mode

The guiding mode is enabled with the ***DET.ACQi.GUIDING*** keyword. When the guiding mode is enabled then the offset correction vector for secondary auto-guiding is continuously transferred from the referred acquisition process to the database (see Table 14). This takes effect only when the acquisition process actually computes and provides the offset frame.

The keyword may be set to 'T' for one or more processes. All processes which have the guiding mode enabled will export their offset vectors to the guiding database attribute. In praxis usually only one process will have the guide-star within its scope.

Caution:
Enabling the guiding mode does not necessarily require the other frames to be transferred continuously as controlled with the ***DET.ACQi.TRANSFER*** *keyword.*

22. Chopping Mode

The synchronization of the detector read-out with a chopper device is done via the external trigger input of the sequencer (see [AD6] and section 11). Chopper synchronization can be done per phase transition or per read-out or per detector integration cycle (i.e. per "*DIT*"). The chopping mode is enabled or disabled with the setup parameter ***DET.CHOP.ST (T/F)***. This will also enable/disable the external trigger input on all sequencer instances (i.e. the ***DET.SEQi.TRIGGER*** parameters always follow ***DET.CHOP.ST*** but not vice versa). The ***DET.CHOP.ST*** parameter enables/disables the chop-cycle-processing in the data acquisition chain whereas the ***DET.SEQi.TRIGGER*** parameter just enables/disables the "wait-for-trigger" states in the sequencer program.

The chopper transition time can be passed to the sequencer program via the setup parameter ***DET.CHOP.TRANSTIM (seconds)***.

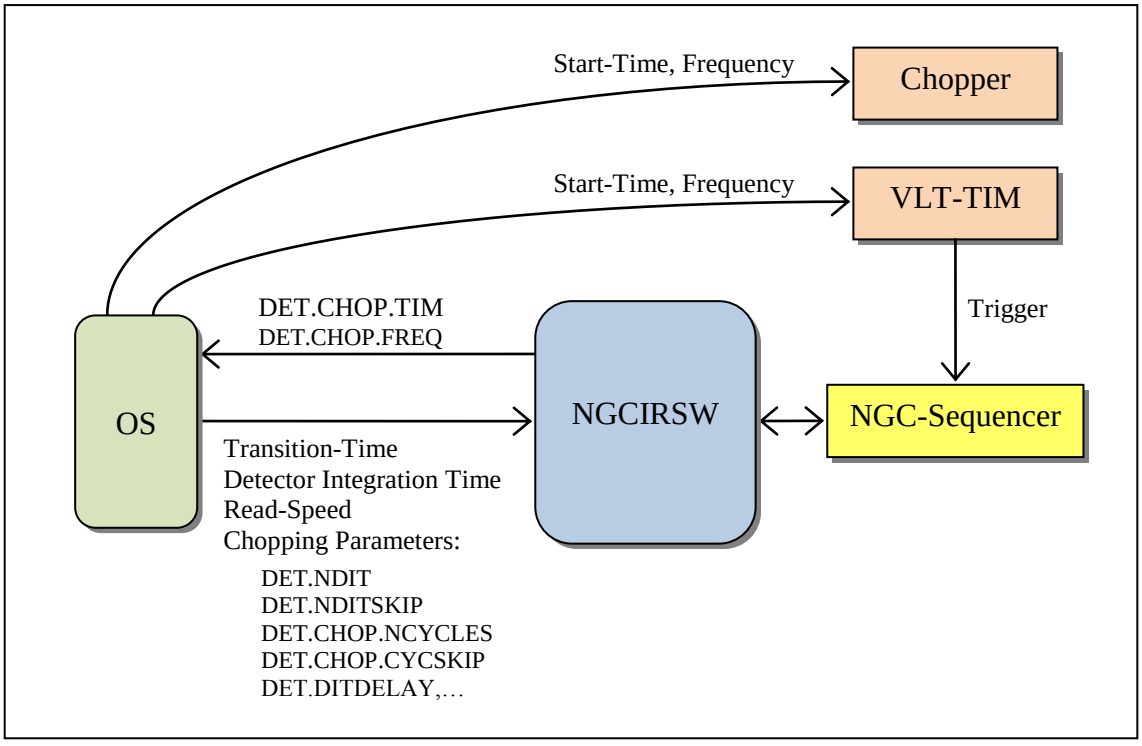

Figure 28 Chopping Mode

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. Then ***DET.NDIT*** + ***DET.NDITSKIP*** integrations will be done on each chopper phase (half cycle). The ***DET.NDITSKIP*** integrations are skipped at the beginning of each half cycle. 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***'.

Caution:
*When **DET.CHOP.ST** is "T" then the chopper and the NGC external trigger must be synchronized. Otherwise the produced data are corrupted. It is still possible to take snapshots in queue-mode (see section 10.1.3) but then the **DET.NDITSKIP** parameter must be set to zero.*

Changes with respect to IRACE:

The handling of the chopping mode is backwards compatible with IRACE. The database point and attributes have changed. Some additional parameters (DET.CHOP.TIM) are available to communicate proper synchronization features.

22.1. Synchronize per Phase-Transition

One trigger-signal is delivered at each chopper phase transition (see Figure 29). 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 28).

The NGC controller electronics can take care, that the sequence always starts in the same phase (e.g. always "*on object*"). This is achieved by alternating the signals on "*trigger input A*" and "*trigger input B*". A re-start of the sequencer program can then only happen upon receiving a signal on "*trigger input A*".

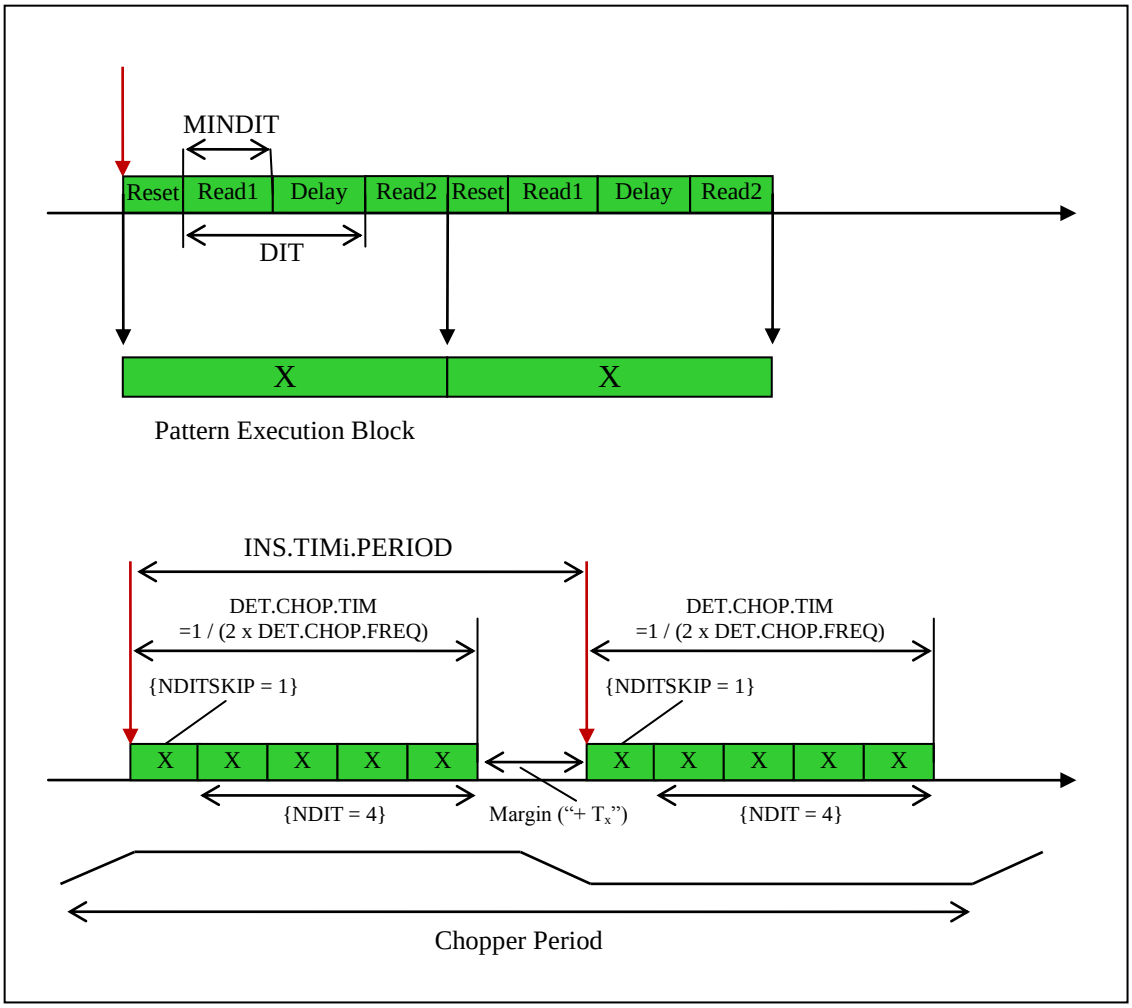

Figure 29 Synchronize per phase-transition

A maximum value for the chopping frequency is computed within the sequencer program and is stored in the parameter ***DET.CHOP.FREQ (Hz)***. The computed value can either be retrieved via the ***STATUS*** command or via the database attribute '***<alias>ngcircon:chopper.freq***' (see Table 16). Accordingly a minimum value for the ***INS.TIMi.PERIOD*** is computed by the sequencer program and is stored in the

parameter ***DET.CHOP.TIM***. The computed value can either be retrieved via the ***STATUS*** command or via the database attribute '***<alias>ngcircon:chopper.tim***' (see Table 16). Typically ***DET.CHOP.TIM = 1 / (2 x DET.CHOP.FREQ)*** but there may be arbitrary usage of these parameters to communicate the built-in synchronization features.

A margin ("+T_x") is needed to compensate rounding errors in the TIM software. When the trigger pulse (red) falls into the pattern execution area (green) then the pulse is missed and only every second trigger comes through with the consequence that integration only takes place on one chopper phase (self subtraction).

Typically we have the relation:

DET.CHOP.TIM = f(DET.SEQi.DIT, DET.NDIT, DET.NDITSKIP, …)

With the constraint:

INS.TIMi.PERIOD >= DET.CHOP.TIM +T_x (with Tx ~ 2-20 microseconds)

The chopper half period is equal to ***INS.TIMi.PERIOD***.

22.2. Synchronize per Read-Out

One trigger signal is delivered per detector read-out (see Figure 30). The signals "*trigger input A*" and "*trigger input B*" are shortened in that case. Typically this mechanism is used in the so-called ***Uncorrelated*** (or "***Rolling***") read-out modes.

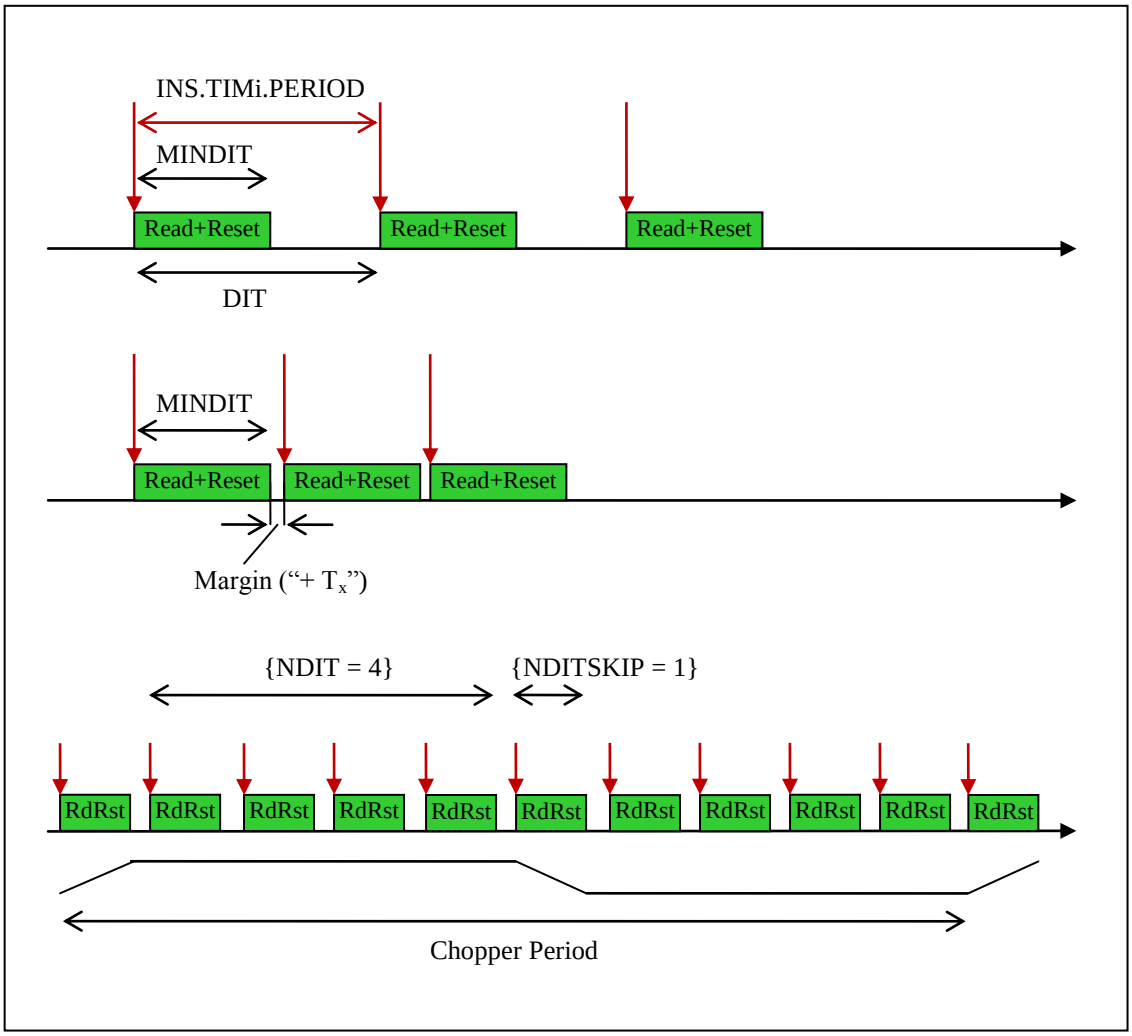

Figure 30 Synchronize per read-out

The detector integration time (DIT) is directly set via the NGC-TIM and therefore it is always equal to the NGC-TIM period (***INS.TIMi.PERIOD***).

When applying the minimum DIT (i.e. DIT=MINDIT) then still a margin ("+ T_x") is needed to compensate rounding errors in the TIM software. This means that the ***INS.TIMi.PERIOD*** must never be set exactly equal to ***DET.SEQi.MINDIT***. When the trigger pulse (red) falls into the pattern execution area (green) then the pulse is missed and only every second trigger comes through. For the acquisition then the whole chopper period falls into ***DET.NDIT*** integrations and the exposure will then take twice as long as expected.

The ***DET.SEQi.DIT*** parameter should always be set to ***INS.TIMi.PERIOD*** in order to finally have the right value in the FITS header. Typically the sequencer program will then also set the ***DET.CHOP.TIM*** parameter to ***DET.SEQi.DIT***. Then there is the following relation between these three parameters.

DET.CHOP.TIM = DET.SEQi.DIT = INS.TIMi.PERIOD.

With the constraint:

INS.TIMi.PERIOD >= DET.SEQi.MINDIT +T_x (with Tx ~ 2-20 microseconds)

The chopper half period must be a multiple of ***INS.TIMi.PERIOD***.

22.3. Synchronize per DIT

One trigger signal is delivered at the start of each detector integration cycle (see Figure 31). The signals "*trigger input A*" and "*trigger input B*" are shortened in that case. Typically this mechanism is used for the ***Double-Correlated*** (or "***CDS***") read-out modes and for the non-destructive modes ("***Least Square Fit***" or "***Fowler Sampling***").

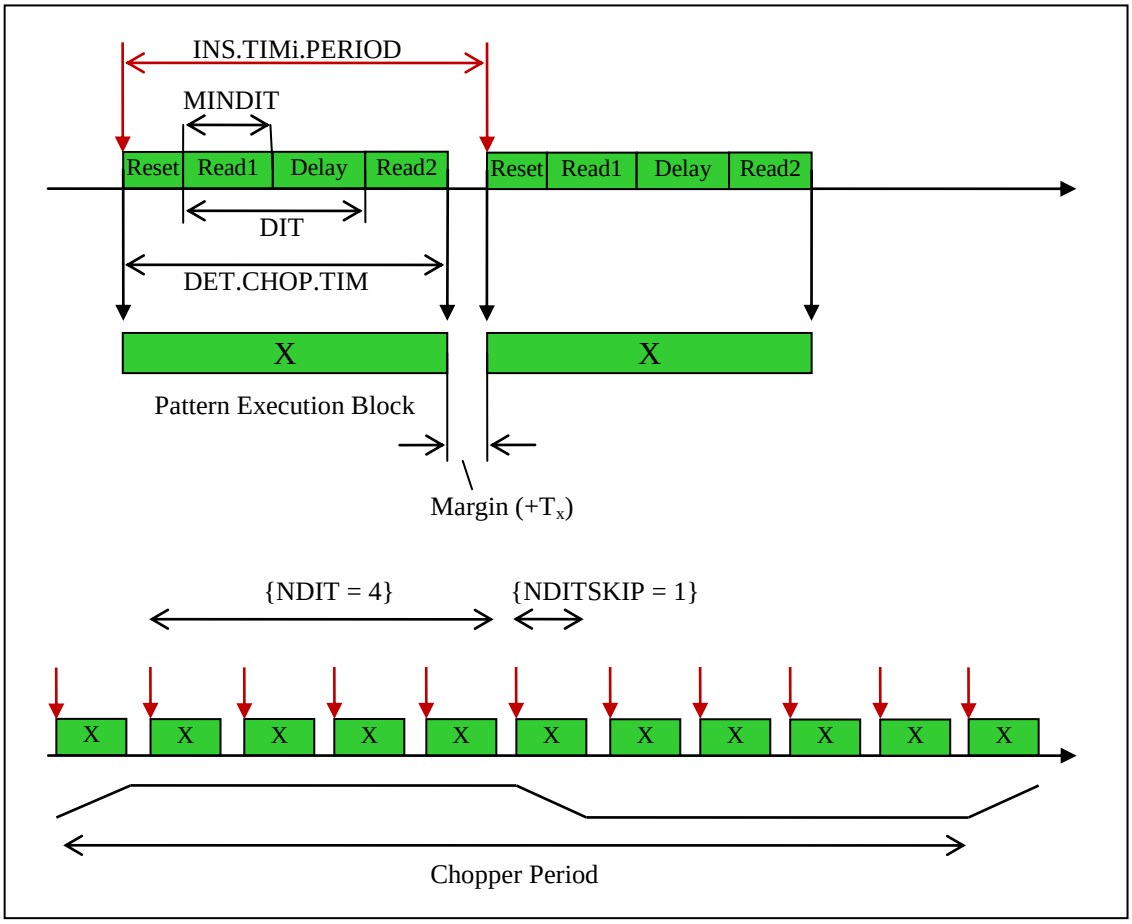

Figure 31 Synchronize per DIT

Here the detector integration time (DIT) is not equal to the NGC-TIM period (***INS.TIMi.PERIOD***). The integration time is set via the ***DET.SEQi.DIT*** parameter which causes the sequencer program to execute delay patterns of appropriate length. Furthermore the second read-out ("*Read2*") and the reset pattern ("*Reset*") add to the length of the pattern execution block ("**X**"). There may be more even more delay patterns (e.g. ***DET.DITDLEAY*** between "*Reset*" and "*Read1*").

The NGC-TIM signal does not trigger the *array read-out* but it triggers the execution of a *pattern execution block*. Again a margin ("+ T_x") is needed to avoid that due to TIM software internal rounding the trigger pulse (red) falls into the pattern execution block (green).

Caution:
The margin refers to the pattern execution block and not to the detector integration time. In this mode it is allowed to set ***DET.SEQi.DIT*** *exactly equal to* ***DET.SEQi.MINDIT****.*

The length of the pattern execution block is computed by the sequencer program and can be retrieved from the NGC-DCS via status command ("***STATUS –function DET.CHOP.TIM***") or from the database attribute "***<alias>ngcircon:chopper.tim***" (see Table 16). It may change whenever a sequencer program parameter changes (e.g. ***DET.SEQi.DIT, DET.DITDELAY, read-speed, …***).

Here we have the relation:

$$DET.CHOP.TIM = f(DET.SEQi.DIT, DET.DITDELAY, read\text{-}speed, \ldots)$$

$$INS.TIMi.PERIOD = DET.CHOP.TIM + T_x \text{ (with Tx ~ 2-20 microseconds)}$$

The chopper half period must be a multiple of ***INS.TIMi.PERIOD***.

23. AO-Head Interface

The AO-head interface provides additional functionality such as various environment sensors, over-illumination protection and the 3-level-clocks. Also the bias-generator differs from the one used for the normal NGC front-end-basic board (FEB).

23.1. Module Creation

Before the interface can be used the AO-head subsystem has to be instantiated in the NGC system configuration file:

```
DET.AOi.DEVIDX    <device index>; # associated device index
DET.AOi.ROUTE     "<route>";     # route to module
DET.AOi.NAME      "<name>";      # module name (optional)
```

Example:

```
DET.AO1.DEVIDX    1;          # associated device index
DET.AO1.ROUTE     "0x2";      # route to module
DET.AO1.NAME      "AO-HEAD"; # module name (optional)
```

This will create one instance of the AO-head interface.

23.2. Sensor Polling

Three sensors are supported per module: a relative humidity sensor, a temperature sensor and a pressure sensor. The pressure sensor is not yet implemented (TBC).

The global status polling must be enabled in order to read out the sensors regularly (set ***DET.CON.POLL*** = "T" either via ***SETUP*** command or in the system configuration file). A sensor in error state will no more be polled unless it had been reset (see Table 47). The sensor values are written into the database (see section 23.10). All sensor values pass through a running median filter (*filter-length* is "9"). The sensor read-out during status polling can explicitly be disabled by setting the parameter ***DET.AOi.SENSENA*** to "F".

23.3. Over-Illumination Protection

A flag indicating that the over-illumination protection has been triggered is read during each polling cycle from AO-head electronics. The value can be read via ***STATUS*** command (***DET.AOi.OVRILLUM***) or from the database. The flag can be reset with the ***AORESET*** command (see Table 47).

Caution:
It is the responsibility of the supervising software to take the proper action (e.g. stop data acquisition/control-loop, issue alarm etc.).

23.4. Over-Temperature Protection

A flag indicating that the over-temperature protection has been triggered is read during each polling cycle from AO-head electronics. The over-temperature protection switches off the main components of the DFE. The system then goes to error state (sub-state "***error***"). This can only be recovered by power cycling the DFE.

23.5. Gain

The gain for the high-voltage clocks can be set via ***SETUP*** command (see Table 49).

The gain is set as an integer number. This value will internally be converted to the corresponding physical voltages to be applied in order to achieve a certain gain. When a new voltage configuration file is loaded or when the clock levels of the high voltage clocks are explicitly set then the actual gain will be overwritten. A setup of the gain will then re-apply the value.

The gain is always reset to 1 when the system goes from ***LOADED/STANDBY*** state to ***ONLINE*** state.

23.6. Phase-Delays and 3-Level-Clocks

The phase-delays and the voltage-levels of the 3-level-clocks can be set via ***SETUP*** command (see Table 49).

The 3-level-clock circuitry can generate four different clock levels. The voltage conversion formula for each clock level is:

```
Register Value = (7.391 * volt) + 128.0
```

The phase-delays are directly set in ticks.

23.7. Analog Front-End Electronics (AFE)

The analog front-end electronics (AFE) of the AO-head can be controlled through a dedicated serial peripheral interface (SPI) which is placed on two communication ports (0x8600 – left side, 0x8601 – right side). The driver interface process provides a virtual address space (0xF000 through 0xFFFF) for the AFE registers and takes care of the proper protocol timing. The speed of the serial interface can be adapted via the ***DET.AOi.AFEDELAY*** keyword. The delay is given in ticks of 5 milliseconds.

The setup-keywords for the AFE control are described in Table 50. A GUI is available for tuning the AFE at run-time (see Figure 36).

The whole interface can be disabled by setting the ***DET.AOi.AFE*** keyword to "F".

23.8. Data Acquisition

23.8.1. Data Links

The data is transmitted via a serial link to the dedicated adaptive optics system. The size of the transmitted data frame can be set with the ***DET.AOi.NX/NY*** keywords. These do only change the size of the frame and not the geometry of the image.

Additionally the data can be acquired via the standard data-link on the NGC-LLCU. The data-link can be enabled and disabled with the ***SETUP*** keyword ***DET.AOi.DATALINK*** (T|F). The default value is 'F' (*disabled*) but may be changed in the controller electronics system configuration file (section 17.1). The ***ngcppAO*** acquisition process can be used to assemble and process images for detector evaluation and tests.

23.8.2. Acquisition Process

Syntax: ***ngcppAO [-map <n>] [-skip <n>] [-setup <n>] [-nosort] [-sort <c1> <c2> <c3> <c4> <c5> <c6> <c7> <c8>]***

The "*-map*" option can be used to assemble more than one frame into a single image. This helps to overcome frame rate limitations. The y-dimension (***DET.ACQi.WIN.NY***) must be a multiple of the number of mapped frames.

The "*-skip*" option will internally enlarge the DMA by <n> frames but process only the last frame of this DMA-transfer. Practically this means that only every *(n+1)th* frame will be processed. Also this helps to overcome frame rate limitations.

The channel sorting can be explicitly set with the "*-sort*" option. E.g. the option "*-sort 1 0 3 2 5 4 7 6*" will swap pairs of channels. The "*-setup*" options selects various pre-defined pixel- and channel sorting schemes:

"*-setup 0*": reshaped boards, channel sorting + {Flip-X, Flip-Y} – this is the default selection
"*-setup 1*": reshaped boards, 1-to-1 channel sorting (i.e. "*-sort 0 1 2 3 4 5 6 7*"), no Flip-X, no Flip-Y
"*-setup 2*": reshaped boards, channel sorting as on SPARTA-Test-Tool, no Flip-X, no Flip-Y
"*-setup 3*": old boards, sorted + {Flip-X, Flip-Y}
"*-setup 4*": old boards, sorted, no Flip-X, no Flip-Y

In all cases the "*-nosort*" option will disable the pixel sorting algorithm. The "*-setup*" option can be redefined at run-time via the ***DET.AOi.SETUP*** parameter (see Table 49). In all cases the "*-sort*" option overrules the channel sorting applied by the "*-setup*" option or by the ***DET.AOi.SETUP*** parameter.

23.9. Commands

AORESET	**-module**	Integer	Specify module index (starts with 1.
	-all	Logical	Reset all sensors.
	-rhum	Logical	Reset the relative humidity sensor.
	-temp	Logical	Reset temperature sensor.
	-pressure	Logical	Reset pressure sensor (dummy function).
	-ovrillum	Logical	Reset over-illumination sensor (dummy function).
	-afe	Logical	Reset analog front end electronics (AFE).

Table 47 AO-Head Commands

Actually the ***AORESET*** command only performs a soft reset. That means that after a sensor went to error state just the soft-state is reset and the sensor is read-out again during the next sensor polling cycle.

Caution:
The "AORESET –all" command includes a reset of the bias-generator. The voltage table has to be re-loaded afterwards.

23.10. Database

One instance ("*ao_<i>*") of the ***ngcdcsAO*** database class (see Table 48) is used per module. Database indexes always start with zero.

Example (*AO-instance 1*): *<alias>ngcdcs:ao_0.<attribute>*

Attribute	Type	Description
name	BYTES64	Module name.
rhum	DOUBLE	Relative humidity (in percent).
rhumError	INT32	Humidity sensor error (this is set to1 in case of an error)
temp	DOUBLE	Temperature (in degree C).
tempError	INT32	Temperature sensor error (this is set to1 in case of an error)
pressure	DOUBLE	Pressure (in kPa).
pressureError	INT32	Pressure sensor error (this is set to1 in case of an error)
gain	INT32	Gain.
dataLink	INT32	Data-link to NGC-LLCU disabled/enabled (0|1).
serialLink	INT32	Serial link to adaptive optics system disabled/enabled (0|1).
overIllumination	INT32	Over-illumination flag (0|1).
master	INT32	Synchronization master flag.
delay	Vector[16] (INT32)	16 delays (in ticks),
viph	Vector[4] (DOUBLE)	3-level-clocks voltage levels (I-phase).
vsph	Vector[4] (DOUBLE)	3-level-clocks voltage levels (S-phase).
nx	INT32	Image transfer x-dimension.
ny	INT32	Image transfer y-dimension.
afeGain	Vector[8] (DOUBLE)	AFE amplifier gain for 8 channels (in dB).
afeClamp	Vector[8] (INT32)	AFE clamp level for 8 channels.
afeInput	Vector[8] (INT32)	AFE input source for 8 channels (0=CCD,1=AUX1,2=AUX2).
afeClampL	INT32	AFE clamp enabled (left side).
afeClampR	INT32	AFE clamp enabled (right side).

Table 48 AO-Head Database Class (ngcdcsAO.class)

23.11. Setup and Status Parameters

The following setup and status keywords have been defined:

Keyword	Type	Description
DET.AOi.REV	Integer	Hardware revision (0 = prototype, 1 = reshaped).
DET.AOi.GAIN	Integer	Gain.
DET.AOi.MAXGAIN	Integer	Maximum gain (default: 2000).
DET.AOi.SENSENA	Logical	Enable/disable sensor read-out during status-polling By default the sensor read-out is enabled.
DET.AOi.DELAYi	String	16 configurable clock delays (in ticks).
DET.AOi.IPHLVLi	Double	3-level-clocks voltage levels (I-phase).
DET.AOi.SPHLVLi	Double	3-level-clocks voltage levels (S-phase).
DETi.AOi.MASTER	Logical	Synchronization master flag.
DET.AOi.DATALINK	Logical	Enable/disable the data-link to the NGC-LLCU.
DET.AOi.SERLINK	Logical	Enable/disable the serial link to the adaptive optics system.
DET.AOi.NX	Integer	Image transfer x-dimension.
DET.AOi.NY	Integer	Image transfer y-dimension.
DET.AOi.SETUP	Integer	Apply a new pixel sorting option (see section 23.8.2).
DET.AOi.EXPABORT	Logical	Flag indicating whether or not a running exposure should be aborted when over-illumination occurs. When set to ‘T’ the exposure is aborted.
DET.AOi.QUADi.HVCALi	Double	Gain calibration voltages for 4 quadrants. The first voltage is the voltage for unity gain 1. The second and third voltage is measured on two different points on the calibration curve.
DET.AOi.QUADi.EQOFFSET	Double	Offset parameter of the calibration equation.
DET.AOi.QUADi.EQSLOPE	Double	Slope parameter of the calibration equation.
DET.AOi.REDELAYi	Double	8 rising edge calibration delays (in ticks).
DET.AOi.FEDELAYi	Double	8 falling edge calibration delays (in ticks).
DET.AOi.AFE	Logical	Enable AFE interface.
DET.AOi.AFEGAINi	Double	AFE amplifier gain for 8 channels (in dB).
DET.AOi.AFECLAMPi	Integer	AFE clamp level for 8 channels.
DET.AOi.AFEINi	Integer	AFE input source for 8 channels (0=CCD,1=AUX1,2=AUX2).
DET.AOi.AFECLMPL	Logical	AFE clamp enabled (left side).
DET.AOi.AFECLMPR	Logical	AFE clamp enabled (right side).
DET.AOi.AFEDELAY	Integer	AFE serial i/o delay (in ticks of 5 milliseconds).

Table 49 AO-Head Setup and Status Keywords

Keyword	Type	Description
DET.AOi.AFE	Logical	Enable AFE interface.
DET.AOi.AFEGAINi	Double	AFE amplifier gain for 8 channels (in dB).
DET.AOi.AFECLAMPi	Integer	AFE clamp level for 8 channels.
DET.AOi.AFEINi	Integer	AFE input source for 8 channels (0=CCD,1=AUX1,2=AUX2).
DET.AOi.AFECLMPL	Logical	AFE clamp enabled (left side).
DET.AOi.AFECLMPR	Logical	AFE clamp enabled (right side).
DET.AOi.AFEDELAY	Integer	AFE serial i/o delay (in ticks of 5 milliseconds).

Table 50 AFE Setup and Status Keywords

Additionally the following status keywords (*read-only*) are supported:

Keyword	Type	Description
DET.AOi.OVRILLUM	Logical	Over-illumination flag. A 'T' value indicates over-illumination.
DET.AOi.RHUM	Double	Humidity sensor value (in percent).
DET.AOi.TEMP	Double	Temperature sensor value (in degree C).
DET.AOi.PRES	Double	Pressure sensor value (in kPa).

Table 51 AO-Head Status Keywords

In case the sensor read-out fails an error attribute is set for this sensor and the ***STATUS*** command will return an error message until the sensor is reset with the ***AORESET*** command (see Table 47). All status keyword values are reported in the FITS header when an exposure is taken with the NGC base software. When the sensor is in error state the corresponding keyword is not written to the FITS-header (TBD).

23.12. Graphical User Interface

The AO-interface can be operated through the NGC engineering GUI. The control- and status-widgets are located in the notebook area.

Figure 32 AO-head Control Widget

The control widget provides a mean to shift clocks by ticks. The shifted pattern is shown in the result display:

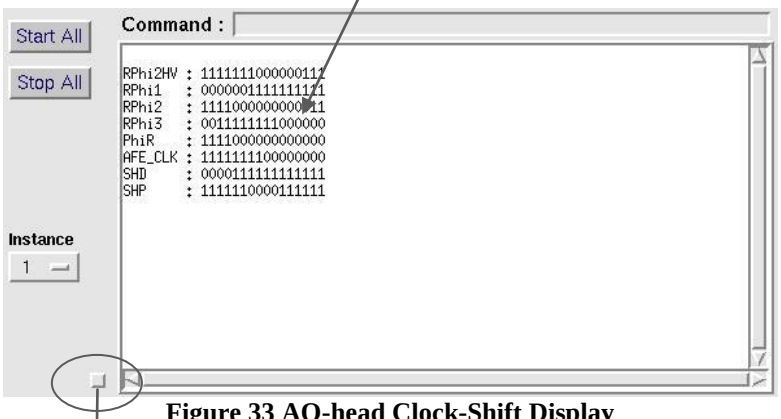

Figure 33 AO-head Clock-Shift Display

Pressing the button on the lower left side of the result widget switches to a high-resolution display:

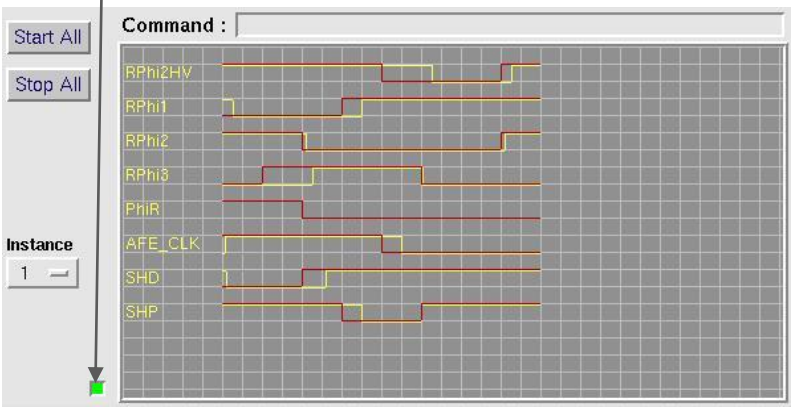

Figure 34 AO-head Clock-Shift Display (High-Res)

The AO-head sensor values are shown in the *AOSENS*-tab.:

Figure 35 AO-head Sensor Widget

The sensor values are updated at each polling cycle when status-polling is switched on.

A third tab in the notebook controls the analog front-end electronics (*AFE*) of the AO-head:

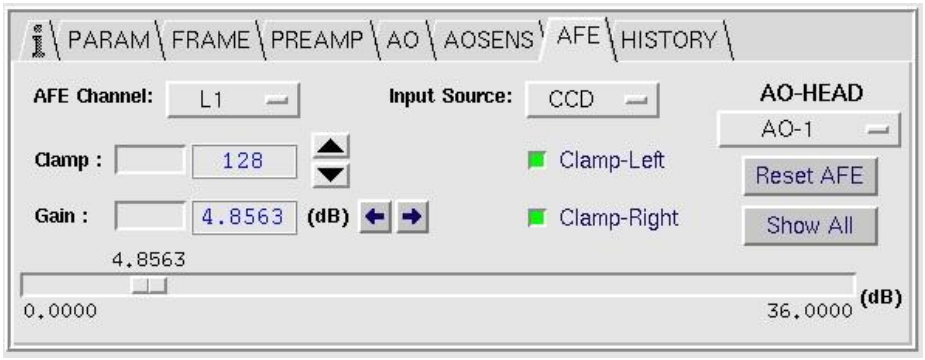

Figure 36 AO-head AFE Control Widget

23.13. Gain Calibration

A routine is provided which automatically calibrates the high voltage clocks by determining and setting the following parameters for each of the four sets of high voltage clocks (i.e. CCD quadrants):

```
DET.AOi.QUAD1.HVCAL2   # HV clock voltage for gain of 10   (20db)
DET.AOi.QUAD1.HVCAL3   # HV clock voltage for gain of 1000 (60db)
DET.AOi.QUAD2.HVCAL2   # HV clock voltage for gain of 10   (20db)
DET.AOi.QUAD2.HVCAL3   # HV clock voltage for gain of 1000 (60db)
DET.AOi.QUAD3.HVCAL2   # HV clock voltage for gain of 10   (20db)
DET.AOi.QUAD3.HVCAL3   # HV clock voltage for gain of 1000 (60db)
DET.AOi.QUAD4.HVCAL2   # HV clock voltage for gain of 10   (20db)
DET.AOi.QUAD4.HVCAL3   # HV clock voltage for gain of 1000 (60db)
```

The routine finds the voltage setting for each of the four high voltage clocks that corresponds to gains of 10 and 1000 and updates these values in a configuration file.

23.13.1. Procedure

First the user sets the illumination such that a ~ 5000ADUs illuminated image is obtained in a 1 second exposure at unity gain. Flat field is best but other illumination patterns could be entertained. All data is taken through the NGC data-link. The steps are as follows:

1) Set binning to *1x1* and image size to *240x240*.
2) Record100 images at unity gain and 1 ms exposure. Store the average (called subsequently the "*bias image*") for later use and write it to disk for diagnostic purposes (*CalBias.fits*).
3) Record 10 images at unity gain and 1 sec exposure. Calculate the average image and subtract the "*bias image*". Store the resultant image (called subsequently "*x1 image*") for later use and write it to disk for diagnostic purposes (*CalX1.fits*).
4) Set the clock voltage of each quadrant to the respective value defined by the keyword ***DET.AOi.QUADi.HVCAL2***.
5) Record 100 images at 100 ms exposure. Calculate the average image and subtract the "*bias image*" (determined in step 2). Store the resultant image (called subsequently "*x10 image*") and write to disk for diagnostic purposes (*CalX10.fits*).
6) Divide "*x10 image*" "by "*x1 image*". Calculate the mean or median for each of the four quadrants. How close the value is to 1, depends on how close the gain was set to 10. Subtract this value from 1 and multiply by 10 times the "*gain of 10 sensitivity parameter*" (***DET.AOi.QUADi.HVSTY2***) in units of V/gain ratio in order to determine how much to increment or decrement the clock voltage. Increment/decrement the clock voltage by this value.

7) Repeat steps 5 and 6 until the gain is within a defined tolerance of the required final value. The tolerance set through the keyword ***DET.AOi.QUADi.HVPRC2***. The default precision is 1%. Take note of the final clock voltage values for each quadrant and write the value into ***DET.AOi.QUADi.HVCAL2*** in the configuration file.

8) Set the clock voltage of each quadrant to the respective value defined by ***DET.AOi.QUADi.HVCAL3***.

9) Record 500 images at 1 ms exposure. Calculate the average image and subtract the "*bias image*" (determined in step 2). Store the resultant image (called subsequently "*x1000 image*") for later use and write to disk for diagnostic purpose (*CalX1000.fits*).

10) Divide "*x1000 image*" by "*x1 image*". Calculate the mean or median for each of the four quadrants. How close the value is to 1, depends on how close the gain was set to 1000. Subtract this value from 1 and multiply by 1000 times the "gain of 1000 sensitivity parameter" (***DET.AOi.QUADi.HVSTY3***) in units of V/gain ratio to determine how much to increment or decrement the clock voltage. Increment/decrement the clock voltage by this value.

11) Repeat steps 9 and 10 until the gain is within a define tolerance of the required final value. The tolerance set through the keyword ***DET.AOi.QUADi.HVPRC3***. The default precision is 1%. This value could be different from value computed in step 7. Take note of the final clock voltage for each quadrant and write the value into ***DET.AOi.QUADi.HVCAL3*** in the configuration file.

If over-illumination or error occurs, then the routine halts indicating what has happened. It is possible to define a map of pixels to be included in the analysis (***DET.AOi.PIXMASK***). The pixel map supports all FITS-files with integer data format BITPIX=8, 16 or 32. Invalid pixels have a zero value (#0). Valid pixels have a non-zero value (preferably #1). Two thresholds (***THRMIN***, ***THRMAX*** – see Table 52) can be used to exclude pixels below/above certain values in the "*X1 image*".

It is possible to determine only the *x10* or *x1000* gain calibration point or to determine calibration points only for certain quadrants. Table 52 contains a list of all options. When the maximum number of iterations is set to 1 (***MAXLOOP = 1***, Table 52) then just the current precision is shown/logged and no calibration is done.

23.13.2. Implementation

The procedure is implemented by making use of the exposure driver concept as described in section 9.10. The name of the exposure driver is "*aocal*". The driver executes the procedural steps as described in the previous section. The calibration can be done

a) sending the ***EXPDRV*** command to DCS (section 9.10.2):

```
EXPDRV -select aocal -fb mon -cfg [param1 value1 … paramN valueN]
EXPDRV -start
EXPDRV -abort
```

b) using the ***ngcdcsExpDrv*** tool (section 9.10.3):

```
ngcdcsExpDrv aocal [-gui] -cfg [param1 value1 … paramN valueN]
```

An index may be appended to the term "*aocal*" (e.g. "*aoca13*") in order to access a certain AO-head instance.

The "*aocal*" exposure driver accepts the configuration parameters as defined in Table 52.

Keyword	***Type***	***Scope***	***Description***
POINT	Integer	Driver	Determine only the given gain calibration point. Supported values are 10 and 1000. A zero value indicates that the calibration will be done for both points.
QUADRANT	Integer	Driver	If the value is greater than zero then the calibration is performed only in the given quadrant.
MAXLOOP	Integer	Driver	Maximum number of calibration iterations.
TARGET	String	Driver	Name of the target file where the configuration keywords will be written after sequence completion. Unless an absolute path is given the file will be written to the directory: `$INS_ROOT/$INS_USER/COMMON/CONFIGFILES/` When the file does not yet exist then it will be created. Otherwise the configuration keywords in the file are updated or appended if not yet present. An empty string or the string "*none*" disables the configuration file output.
LOG	String	Driver	Name of a log file where the current configuration and the results are appended. Unless an absolute path is given the file will be written to the directory `$INS_ROOT/$INS_USER/MISC/`
FILTER	String	Driver	Set calibration filter. Supported names are "*MEAN*" or "*MEDIAN*".
SKIPTIME	Double	Driver	Settling time (in seconds) after restart. Frames will be skipped for that period.
THRMIN THRMAX	Float	Driver	Threshold upper and lower rail. The value is given in ADUs in floating point format. Any value equal to or less than 0.0 will disable the respective threshold.

Keyword	***Type***	***Scope***	***Description***
PIXMASK	String	Server	File name of a pixel-mask to be applied. The file is expected to be a FITS-file marking valid pixels with '1' and invalid pixels with '0'. Supported FITS data-types are integer values with BITPIX = 8, 16 or 32. Unless an absolute path is given the file is read from the directory: $INS_ROOT/$INS_USER/SYSTEM/COMMON/CONFIGFILES An empty string or the string "*none*" disables the pixel mask.
MINPIX	Integer	Driver	Minimum number of valid pixels per quadrant.
HVPRC HVPRC2 HVPRC3 QUADi.HVPRC2 QUADi.HVPRC3	Float	Server	Set the required precision for quadrant <i>. The value is given in percent. If the QUADi index is omitted then the value applies to all four quadrants. HVPRC2 refers to the calibration point 2 (gain x10). HVPRC3 refers to the calibration point 3 (gain x1000).
HVSTY HVSTY2 HVSTY3 QUADi.HVSTY2 QUADi.HVSTY3	Float	Server	Set calibration sensitivity for quadrant <i>. The value is given in the unit [V/gain]. If the QUADi index is omitted then the value applies to all four quadrants. HVSTY2 refers to the calibration point 2 (gain x10). HVSTY3 refers to the calibration point 3 (gain x1000).
DV, DV2, DV3	Float	Driver	Maximum voltage change during calibration. DV2 refers to the calibration point 2 (gain x10). DV3 refers to the calibration point 3 (gain x1000).

Table 52 AO Gain Calibration Keywords

Scope "Driver" means that the parameter is private to the driver. Scope "Server" means that the parameter is imported from the server: when the driver changes the value then the value in the server will also be updated and when the driver exits then the values are kept in the server. The "***DET.AOi.***" categories can be given ahead of each keyword but will be discarded (but the keyword is only taken into account when the index matches the AO-head instance number).

23.14. The AO-Bias Generator

The bias generator for the AO-head is a derivative of the standard ***CLDC***-module. The register i/o and the voltage transfer functions have been adapted to serve the AO-head specific components and circuitries. The bias generator employs 21 bias voltages and additional 7 substrate voltages. The substrate voltages are used to offset the rest of the bias voltages. Each of the bias and substrate voltages has its own voltage transfer function. The bias generator software module is created like a normal ***CLDC***-module as shown in section 17.1.2. An additional type indicator (***DET.CLDCi.SPEC***) tells the system to create the ***AOBIAS***-variant:

Example:

```
DET.CLDC1.DEVIDX  1;           # associated device index
DET.CLDC1.ROUTE  "2";          # route to module
DET.CLDC1.MAXDC   27;          # maximum number of biases
DET.CLDC1.NAME   "AO-Bias-1"; # module name (optional)
DET.CLDC1.SPEC   "AOBIAS";     # module specification
```

The names, values and ranges for the bias- and substrate-voltages can be set via the normal voltage configuration file (see section 16.4.1). When no voltage configuration file is given then a default configuration with the nominal voltages and the maximum possible ranges is applied (see Table 53). The voltages can be changed via ***SETUP*** command in the usual way (**DET.CLDCi.DC1-27**).

Caution:
*When a substrate voltage is changed via **SETUP** command then the associated bias voltages will shift according to their voltage transfer functions. When a voltage configuration file is loaded then the 7 substrate voltages are always applied first.*

Bias Number	Name	Nominal Value	Description
DC1	VHV-H1	1.83 V	High voltage clock upper rail OS5/OS6
DC2	VHV-L1	1.05 V	High voltage clock lower rail OS5/OS6
DC3	VHV-H2	1.83 V	High voltage clock upper rail OS1/OS2
DC4	VHV-L2	1.05 V	High voltage clock lower rail OS1/OS2
DC5	VHV-H3	1.83 V	High voltage clock upper rail OS7/OS8
DC6	VHV-L3	1.05 V	High voltage clock lower rail OS7/OS8
DC7	VHV-H4	1.83 V	High voltage clock upper rail OS3/OS4
DC8	VHV-L4	1.05 V	High voltage clock lower rail OS3/OS4
DC9	SG	10.0 V	Shutter gate
DC10	VR0-H	10.0 V	Serial upper rail
DC11	VR0-L	10.0 V	Serial lower rail
DC12	V0C-H	6.96 V	Clamp upper rail
DC13	V0C-L	6.96 V	Clamp lower rail
DC14	R0DC	0.0 V	Multiplication DC
DC15	VOG	-0.3 V	Output gate
DC16	SDH	0.0 V	Shutter drain high
DC17	SDL	0.0 V	Shutter drain low
DC18	V0R-H	6.37 V	Reset upper rail
DC19	V0R-L	-3.76 V	Reset lower rail
DC20	VOD	25.0 V	Output drain
DC21	VRD	14.4 V	Reset drain
DC22	VSUB	7.0 V	Substrate voltage 1
DC23	VSUB0	2.0 V	Substrate voltage 2
--	VSUB1	7.0 V	Substrate voltage 3 (hardwired with VSUB)
DC24	VSUB2	0.78 V	Substrate voltage 4
DC25	VSUB3	-1.38 V	Substrate voltage 5
DC26	VSUB4	5.0 V	Substrate voltage 6
DC27	VCOM	-0.2 V	Substrate voltage 7
DC28	LED	0.0 V	LED voltage

Table 53 AO-Head Bias- and Substrate-Voltages

24. Infrared Waveform Sensor

The following sections refer to the interface to the SELEX SAPHIRA infrared waveform sensor and fringe tracker [RD15].

24.1. Module Creation

Before the interface can be used to program the SELEX-device it has to be enabled in the NGC system configuration file:

```
DET.SEQi.SLX.ENA    "T|F";          # SELEX interface enabled
```

Example:

```
DET.SEQ1.DEVIDX     1;              # associated device index
DET.SEQ1.ROUTE      "0x2";          # route to module
DET.SEQ1.NAME       "SELEX-SEQ";    # module name (optional)
DET.SEQi.SLX.ENA    "T";            # SELEX interface enabled
```

This will create one instance of the sequencer with the SELEX serial interface enabled.

The serial interface uses the normal clock-lines of the sequencer module (see RD1). In order to make these clock lines available again for general purposes the interface must explicitly be disabled:

```
setup DET.SEQi.SLX.ENA F
```

Caution:
When the SELEX-device is programmed from within the sequencer program (section 24.3.1) then the interface is automatically enabled.

24.2. Windowing Features

The SELEX-device provides full-frame or multiple sub-frame windowing readout modes. The *reset-* and *data-readout-windows* can be defined separately (see [RD15] for details). The frame dimension is 10 columns times 256 rows where each column is 32 pixels wide (see Figure 37). The columns and rows can be enabled and disabled individually by setting the corresponding bit in the window data register (WDR) and in the window reset register (WRR). The programming is done through the serial interface as described in section 24.3.

The column-bits can be set directly via logical parameters. The bit-setting for the 256 rows is not practicable in combination with a keyword based interface. When using the keywords as described in sections 24.3.1.2 and 24.3.2 then the rows are programmed as row-sections defined by "*START*" (pixel offset starting with "1") and "*SIZE*" (row width in pixels). The interface allows up to 48 row-sections to be defined.

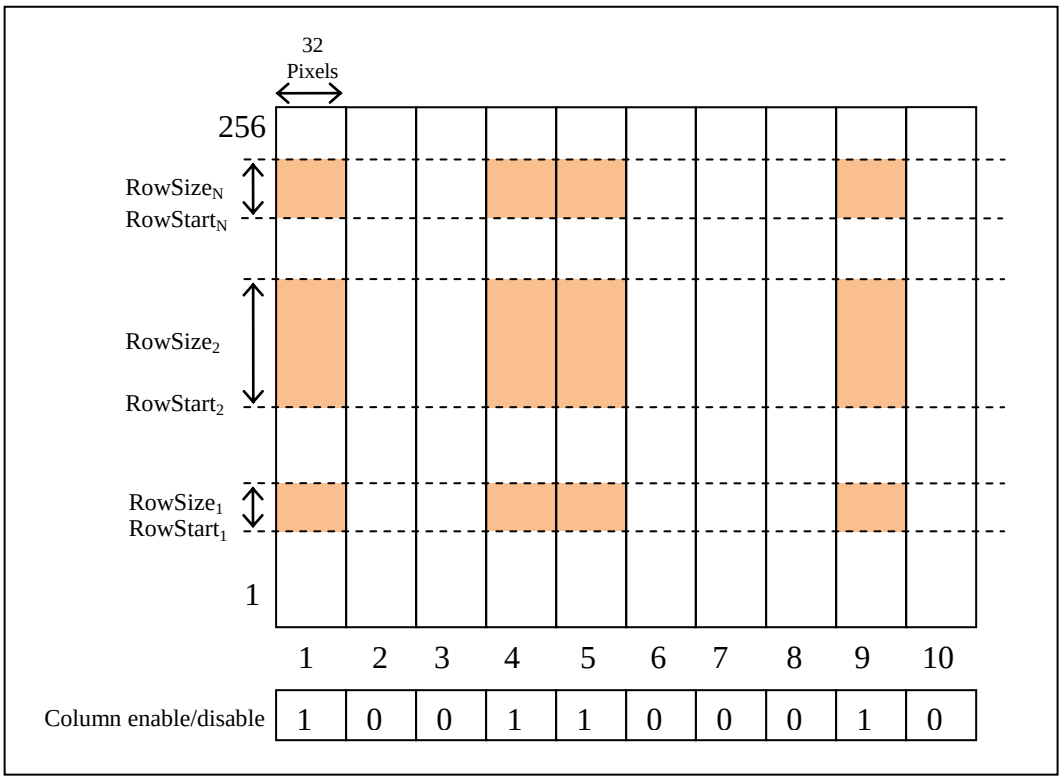

Figure 37 Readout Topology

The sub-windows are glued together by the acquisition processes (section 24.5). The resulting frame size of the assembled image is automatically computed and exported to the NGC acquisition modules.

24.3. The Serial Link Interface

The SELEX-device is configured through an SPI compatible serial data interface (see [RD15] for details). The SPI control signals are connected to the normal sequencer clock lines (see [RD1]) once the interface is enabled as described in section 24.1.

The interface is represented in the detector front-end electronics by a *9x32*-bit shift-RAM. The shift direction is from MSB to LSB. The MSB is placed at the shift-RAM base address. The address is incremented every 32-bits.

24.3.1. Sequencer Program Interface

The SELEX-device can be configured from within the sequencer program. The program loader recognizes the instructions and automatically enables the serial interface whenever such a sequencer program is loaded. The serial interface will remain enabled until it is explicitly disabled again by either loading a sequencer program requesting this (e.g. via a special keyword as described in section 24.3.1.2) or by sending the equivalent setup command ("*setup SET.SEQi.SLX.ENA F*").

24.3.1.1. Direct Shift Register Programming

To directly program the serial shift register from within the sequencer program a set of special keywords (Table 54) has been added to the scripting language instruction set:

```
SCRIPT
…
set svar(SEQ.SHIFTi.WORDi) <hexadecimal-value>
set svar(SEQ.SHIFTi.SIZE)  <number of bits to shift out>
…
SCRIPT_END
```

The ***SEQ***-category has been introduced to indicate that the scope of these keywords does not go beyond the sequencer module shift register.

One *SEQ.SHIFTi*-block refers to one shift instruction. The *SEQ.SHIFTi.WORDi* keywords each contain 32 serial bits where the LSB is placed in the first bit of $WORD_1$ and the MSB is placed in $WORD_N$. The total size in bits is given by the *SEQ.SHIFTi.SIZE* keyword.

Example:

```
set svar(SEQ.SHIFT1.WORD1) 01020304
set svar(SEQ.SHIFT1.WORD2) 0a0b0c0d
set svar(SEQ.SHIFT1.WORD3) 05060708
set svar(SEQ.SHIFT1.WORD4) 10203040
set svar(SEQ.SHIFT1.SIZE) 128
set svar(SEQ.SHIFT2.WORD1) 0000aabb
set svar(SEQ.SHIFT2.SIZE) 16
set svar(SEQ.SHIFT3.WORD1) 000000c1
set svar(SEQ.SHIFT3.SIZE) 8
```

Caution:
This setup passes through without touching the global setup keywords (section 24.3.2).

24.3.1.2. Keyword Setup

To explicitly program the SELEX-device from within the sequencer program a set of special keywords (Table 54) has been added to the scripting language instruction set:

```
SCRIPT
…
set svar(SLX.keyword) value
…
SCRIPT_END
```

Example:

```
set svar(SLX.MODE) "itr"
set svar(SLX.REFPIX) "T"
set svar(SLX.WIN) "T"
set svar(SLX.COL1) "T"
set svar(SLX.COL2) "T"
set svar(SLX.COL9) "T"
set svar(SLX.COL10) "T"
set svar(SLX.ROW1) "1,4"
set svar(SLX.ROW2) "32,8"
```

The ***SLX***-category has been introduced to indicate that the scope of these keywords is private to the SELEX-device. They are only valid within the sequencer program and they are not part of the global interface to the detector sub-system. Nevertheless loading such a sequencer program will supersede and update the global parameters (section 24.3.2).

Keyword	Type	Description
SLX.ENA	Logical	Enable/disable the serial link interface.
SLX.MODE	String	SELEX readout mode.
SLX.MCR	Integer	Set multiplexer control register. The value can be given as binary (e.g. 0b01001100) or hexadecimal (0x…) value. The setting of the first two bits is ignored (hard coded 32-channel mode).
SLX.REFPIX	Logical	Enable/disable reference pixels. Disabling the reference pixels automatically disables the columns 1 and 10.
SLX.EXTRST	Logical	Enable/disable external reset.
SLX.CBENA	Logical	Enable/disable ballast capacitors.
SLX.CBREFENA	Logical	Enable/disable ballast capacitors (reference).
SLX.LSPENA	Logical	Enable/disable line synchronization pulse output.
SLX.CLAMP	Logical	Enable/disable pixel array clamp function.
SLX.WIN	Logical	Enable/disable window read-out. When set to “F” then always the full frame is read.
SLX.CLRWIN	Logical	When set to “T” the window configuration is cleared (no window defined). When no window is defined then also the full frame is read.
SLX.COLi	Logical	Enable the given column (column index ranges from 1 to 10).
SLX.ROWi	String	Row section. Format is: “*start,size*”.
SLX.DCOLi	Logical	Enable the given data column (column index ranges from 1 to 10).
SLX.DROWi	String	Data row section. Format is: “*start,size*”.
SLX.RCOLi	Logical	Enable the given reset column (column index ranges from 1 to 10).
SLX.RROWi	String	Reset row section, Format is: “*start,size*”.

Table 54 SELEX Sequencer Program Keywords

24.3.2. Parameter Interface

To program the SELEX-device from within the scope of the interface to the detector sub-system a set of ***DET***-category keywords has been added. The setup parameters can be specified in the NGC system configuration file. The parameters will not be applied when the serial interface is disabled (i.e. *DET.SEQi.SXL.ENA = F*).

Keyword	Type	Description
DET.SEQi.SLX.ENA	Logical	Enable/disable the serial link interface.
DET.SEQi.SLX.MODE	String	SELEX readout mode.
DET.SEQi.SLX.MCR	Integer	Set multiplexer control register (hexadecimal value). The setting of the first two bits is ignored (hard coded 32-channel mode).
DET.SEQi.SLX.REFPIX	Logical	Enable/disable reference pixels. Disabling the reference pixels automatically disables the columns 1 and 10.
DET.SEQi.SLX.EXTRST	Logical	Enable/disable external reset.
DET.SEQi.SLX.CBENA	Logical	Enable/disable ballast capacitors.
DET.SEQi.SLX.CBREFENA	Logical	Enable/disable ballast capacitors (reference).
DET.SEQi.SLX.LSPENA	Logical	Enable/disable line synchronization pulse output.
DET.SEQi.SLX.CLAMP	Logical	Enable/disable pixel array clamp function.
DET.SEQi.SLX.WIN	Logical	Enable/disable window read-out. When set to “F” then always the full frame is read.
DET.SEQi.SLX.CLRWIN	Logical	When set to “T” the window configuration is cleared (no window defined). When no window is defined then also the full frame is read.
DET.SEQi.SLX.COLi	Logical	Enable the given column (column index ranges from 1 to 10).
DET.SEQi.SLX.ROWi	Logical	Section row-start offset in pixels (coordinates start with 1).
DET.SEQi.SLX.NROWi	Logical	Number of rows in this section.
DET.SEQi.SLX.DCOLi	Logical	Enable the given data column (column index ranges from 1 to 10).
DET.SEQi.SLX.DROWi	Logical	Data section row-start offset in pixels (coordinates start with 1).
DET.SEQi.SLX.NDROWi	Logical	Number of data rows in this section.
DET.SEQi.SLX.RCOLi	Logical	Enable the given reset column (column index ranges from 1 to 10).
DET.SEQi.SLX.RROWi	Logical	Reset section row-start offset in pixels (coordinates start with 1).
DET.SEQi.SLX.NRROWi	Logical	Number of reset rows in this section.

Table 55 SELEX Setup and Status Keywords

24.4. Built-In Pre-Processor

The 10MHz version of the NGC 32-channel ADC-board [RD1] includes a built-in pixel pre-processor. The pre-processor supports pixel integration (add-up N frames) and sub-pixel sampling (add up N samples per pixels). The maximum frame size is *320x256* pixels which exactly matches the dimension of the SELEX-device. Smaller frame sizes suitable for sub-window readouts are configurable through setup keywords (see Table 56).

Keyword	Type	Description
DET.ADCi.PPENA	**Logical**	Enable/disable the built-in pre-processor.
DET.ADCi.PPNX	**Integer**	Frame x-dimension. A zero frame dimension disables the pre-processor.
DET.ADCi.PPNY	**Integer**	Frame y-dimension. A zero frame dimension disables the pre-processor.
DET.ADCi.PPNINT	**Integer**	Pre-processor integration frame counter. This defines the number of frames to be added up.
DET.ADCi.PPNSAMP	**Integer**	Number of pre-processor sub-pixel samples.
DET.ADCi.PPFOWLER	**Logical**	Enable/disable Fowler mode (**DET.ADCi.PPNINT** Fowler pairs).
DET.ADCi.PPINVERT	**Logical**	Invert Fowler pairs.

Table 56 Pre-processor Keywords

Caution:
When the pre-processor is enabled and pixel-adding is done (DET.ADCi.PPNINT > 1 or DET.ADCi.PPNSAMP > 1) then the pixel word width is 32 bits instead of 16 bits. The pre-processor always delivers sums (32-bit signed integer) and not normalized mean-values. When Fowler mode is enabled then the pixel word width is always 32 bits.

The parameters are also available in the ADC module database class (see section 6.3).

When the parameter ***DET.NSAMPPIX*** is defined within the actual scope then it is directly forwarded to all ***DET.ADCi.PPNSAMP*** parameters.

When the parameter ***DET.NDSAMPLES*** is defined within the actual scope then {***DET.NDSAMPLES / 2***} is directly forwarded to all ***DET.ADCi.PPNINT*** parameters.

The pre-processor can be configured via the NGC engineering GUI:

Figure 38 Built-In Pre-Processor

Caution:

Unless the actual ADC board type is AQ32-10MHz the entered values are ignored.

24.5. Acquisition Processes

24.5.1. Non Pre-Processed Modes

The following modes can be used when the built-in pre-processor (section 24.4) is disabled.

24.5.1.1. Uncorrelated

Syntax: ***ngciracqSLX1***

Parameters:

- *DET.NDIT* (number of frames to be added up)
- *DET.NDITSKIP* (number of frames to skip before adding up)

Data Products:

- **DIT-frame** (one frame per readout). Data type is 16-bit signed integer with BZERO=32768.
- **INT-frame** (mean value of *DET.NDIT* readouts). Data type is 32-bit floating point when *DET.NDIT* > 1, otherwise data type is 32-bit signed integer.
- **STDEV-frame** (standard deviation of *DET.NDIT* readouts). Data type is 32-bit floating point.

24.5.1.2. Double Correlated (Reset-Read-Read)

Syntax: ***ngciracqSLX2***

Parameters:

- *DET.NDIT* (number of DIT-frames to be added up)
- *DET.NDITSKIP* (number of DIT-frames to skip before adding up)

Data Products:

- **DIT-frame** (one frame per two readouts). Data type is 32-bit signed integer.
- **INT-frame** (mean value of *DET.NDIT* DIT-frames). Data type is 32-bit floating point when *DET.NDIT* > 1, otherwise data type is 32-bit signed integer.
- **STDEV-frame** (standard deviation of *DET.NDIT* DIT-frames). Data type is 32-bit floating point.

24.5.1.3. Double Correlated (Read-Reset-Read)

Syntax: ***ngciracqSLX21***

Parameters:

- *DET.NDIT* (number of frames to be added up)
- *DET.NDITSKIP* (number of frames to skip before adding up)

Data Products:

- **DIT-frame** (on frame per two readouts). Data type is 32-bit signed integer.
- **INT-frame** (mean value of *DET.NDIT* DIT-frames). Data type is 32-bit floating point when *DET.NDIT* > 1, otherwise data type is 32-bit signed integer.
- **STDEV-frame** (standard deviation of *DET.NDIT* DIT-frames). Data type is 32-bit floating point.

24.5.1.4. Fowler Sampling

Syntax: ***ngciracqSLX4Fo***

Parameters:

- *DET.NDSAMPLES* (number of frames per integration ramp, must be a multiple of 2)
- *DET.NDSKIP* (number of frames to skip at beginning of each integration ramp)
- *DET.NDIT* (number of integration ramps to be added up)
- *DET.NDITSKIP* (number of integration ramps to skip before adding up)

Data Products:

- **DIT-frame** (one frame per integration ramp). Data type is 32-bit floating point.
- **INT-frame** (mean value of *DET.NDIT* DIT-frames). Data type is 32-bit floating point.
- **STDEV-frame** (standard deviation of *DET.NDIT* DIT-frames). Data type is 32-bit floating point.

24.5.1.5. Fowler Sampling (Pixel Sub-Sampling)

Syntax: ***ngciracqSLX4nFo***

Parameters:

- *DET.NSAMPPIX* (number of samples per pixel)
- *DET.NDSAMPLES* (number of frames per integration ramp, must be a multiple of 2)
- *DET.NDSKIP* (number of frames to skip at beginning of each integration ramp)
- *DET.NDIT* (number of integration ramps to be added up)
- *DET.NDITSKIP* (number of integration ramps to skip before adding up)

Data Products:

- **DIT-frame** (one frame per integration ramp). Data type is 32-bit floating point.
- **INT-frame** (mean value of *DET.NDIT* DIT-frames). Data type is 32-bit floating point.
- **STDEV-frame** (standard deviation of *DET.NDIT* DIT-frames). Data type is 32-bit floating point.

24.5.2. Pre-Processed Modes

The following modes require the built-in pre-processor (section 24.4) to be enabled.

24.5.2.1. Pre-Processed & Uncorrelated

Syntax: ***ngciracqSLXPP1***

Parameters:

- *DET.NDIT* (number of frames added up in the pre-processor)
- *DET.NSAMPPIX* (number of samples per pixel in the pre-processor)

Data Products:

- **INT-frame** (mean value of *DET.NDIT* readouts). Data type is 32-bit floating point when *DET.NDIT* > 1 or *DET.NSAMPPIX* > 1, otherwise data type is 32-bit signed integer.

24.5.2.2. Pre-Processed Multiple Sampling

Syntax: ***ngciracqSLXPP1n***

Parameters:

- *DET.NSAMPPIX* (number of samples per pixel in the pre-processor)
- *DET.NDIT* (number of frames to be added up)
- *DET.NDITSKIP* (number of frames to skip before adding up)

Data Products:

- **DIT-frame** (one frame per readout). Data type is 32-bit floating point when *DET.NSAMPPIX* > 1, otherwise data type is 32-bit signed integer.
- **INT-frame** (mean value of *DET.NDIT* readouts). Data type is 32-bit floating point when *DET.NDIT* > 1 or *DET.NSAMPPIX* > 1, otherwise data type is 32-bit signed integer.
- **STDEV-frame** (standard deviation of *DET.NDIT* readouts). Data type is 32-bit floating point)

24.5.2.3. Pre-Processed Double Sampling

Syntax: ***ngciracqSLXPP2***

Parameters:

- *DET.NSAMPPIX* (number of samples per pixel in the pre-processor)
- *DET.NDSAMPLES* (number of readouts per integration ramp, must be a multiple of 2 assuming that the pre-processor computes 2 sum-frames)
- *DET.NDIT* (number of integration ramps to be added up)
- *DET.NDITSKIP* (number of integration ramps to skip before adding up)

Data Products:

- **DIT-frame** (one frame per integration ramp). Data type is 32-bit floating point when *DET.NSAMPPIX* > 1 or *DET.NDSAMPLES* > 2, otherwise data type is 32-bit signed integer.
- **INT-frame** (mean value of *DET.NDIT* integration ramps). Data type is 32-bit floating point when *DET.NDIT* > 1 or *DET.NSAMPPIX* > 1 or *DET.NDSAMPLES* > 2, otherwise data type is 32-bit signed integer.
- **STDEV-frame** (standard deviation of *DET.NDIT* integration ramps). Data type is 32-bit floating point.

24.5.2.4. Pre-Processed Fowler Sampling

Syntax: ***ngciracqSLXPP4***

Parameters:

- *DET.NSAMPPIX* (number of samples per pixel in the pre-processor)
- *DET.NDSAMPLES* (number of readouts per integration ramp, must be a multiple of 2 assuming that the pre-processor computes {*DET.NDSAMPLES / 2*} Fowler pairs)
- *DET.NDIT* (number of integration ramps to be added up)
- *DET.NDITSKIP* (number of integration ramps to skip before adding up)

Data Products:

- **DIT-frame** (one frame per integration ramp). Data type is 32-bit floating point when *DET.NSAMPPIX* > 1 or *DET.NDSAMPLES* > 2, otherwise data type is 32-bit signed integer.
- **INT-frame** (mean value of *DET.NDIT* integration ramps). Data type is 32-bit floating point when *DET.NDIT* > 1 or *DET.NSAMPPIX* > 1 or *DET.NDSAMPLES* > 2, otherwise data type is 32-bit signed integer.
- **STDEV-frame** (standard deviation of *DET.NDIT* integration ramps). Data type is 32-bit floating point.

25. General Purpose Control Server

The ***ngcdcs*** base SW module provides a control server instance ***ngcdcsEvh*** ("*ngcdcs Event Handler*") for general purposes (e.g. for hardware development and tests). This instance can be used to implement command driven sub-systems to control the NGC detector front-end electronics (see the special configurations in section 17.3.1).

The ***ngcdcsEvh*** server is compatible to ***ngcircon*** except that it does not implement the infrared specific tasks (e.g. chopper control). Both ***ngcdcsEvh*** and ***ngcircon*** use the same server base class (***ngcdcsEVH***).

The command line options, the system startup and shutdown procedures (section 3) and the command interface (section 4) are applicable in the same way as with the ***ngcircon*** server.

The database branch is defined in ***ngcdcs.db***. The macros ***ngcdcsINSTANCE*** and ***ngcdcsROOT*** for the database branch can be defined accordingly as described in section 6.

The default database point is "***<alias>ngcdcs***" (i.e. the default value of ***ngcdcsINSTANCE*** is "***ngcdcs***"). The basic structure of the database is as follows (TBD):

```
--o <alias>ngcdcsINSTANCE --|--o seq_<i>     (sequencer parameters)
                            |--o cldc_<i>    (CLDC parameters)
                            |--o adc_<i>     (ADC module parameters)
                            |--o preamp_<i>  (preamplifier module parameters)
                            |--o acq_<i>     (acquisition module parameters)
                            |--o system      (NGC system parameters)
                            |--o exposure    (exposure parameters)
                            |--o mode        (read-out mode parameters)
```

The database classes for the implemented branches are the same as described in section 6.

26. Server Extensions

Server extensions may be needed for executing application specific code when going to ***ONLINE***- or ***STANDBY***-state, upon data reception (post-processing call-back) and also for the handling of additional ***SETUP*** keywords. This section addresses to software engineers and presupposes knowledge of the C++ programming language and of the ESO VLTSW environment.

26.1. How to create a new Server

A new control server instance is created by deriving from the *NGCIRSW* server base class ***ngcirconEVH***:

```
#include "ngcirconEVH.h"

class xxirconEVH: public ngcirconEVH
{
public:
  // Constructors
  xxirconEVH() {}
  xxirconEVH(ngcdcsCTRL *controller) : ngcirconEVH(controller) {}

  // Destructor
  virtual ~xxirconEVH() {
    Verbose("xxirconEVH: terminating...\n");
    Verbose("xxirconEVH: down...\n");
  }

  // Post-processing call-back
  int PostProcCB(void *, ngcdcs_finfo_t *, eccsERROR *);

protected:

private:
};
```

Then a main process has to be created according to the following template (this is based on the "*ngcircon/templates/xxircon.C*" program):

```
/* Post-processing call-back */
int xxirconEVH::PostProcCB(void *buffer,
                     ngcdcs_finfo_t *finfo,
                     eccsERROR *error)
{
  int ret = ngcbSUCCESS;
  USE(buffer); USE(error);
  Verbose(2, "xxirconEVH: post processor for %s...\n", finfo->name);
  return (ret);
}

/* Control server */
static int serverProcess(int argc, char **argv)
{
  ngcdcsCTRL_CLASS controller;
  xxirconEVH server(&controller);
  int exitStatus;

  // Server default settings
  server.SysCfgDefault("NGCIRSW/ngc.cfg");
  server.DbPointDefault("<alias>ngcircon");

  // Server main-loop
  int exitStatus = ngcdcsServerProcess(&server, argc, argv);

  return (exitStatus);
}

/* Main process */
int main(int argc, char *argv[])
{
  int exitStatus;
  char procName[64];
  ccsERROR error;

  // CCS init
  memset(procName, 0, sizeof(procName));
  ngcdcsSetEnv(argc, argv, procName);
  if (ccsInit(procName, ccsOBI_CARE_OFF|ccsOBI_CLEANUP_ON,
              NULL, ngcdcsEvhKillHandler, &error) == FAILURE)
    {
    errPrint(&error);
    exit(1);
    }

  // Call server
  exitStatus = serverProcess(argc, argv);
  ccsExit(&error);
  exit(exitStatus);
}
```

The following lines should be included in the Makefile of the derived server:

```
DIR = $(shell if [ -r $(INTROOT)/include/ngcdcsMakefile ]; then \
      echo INTROOT; else echo VLTROOT; fi)

include $($(DIR))/include/ngcdcsMakefile
```

Then the compilation flags for the executable server can simply be set to:

```
xxirconEVH_LDFLAGS = $(ngcdcsLDFLAGS)
xxirconEVH_LIBS    = <my libs> ngcirconLib $(ngcdcsLIBS)
```

This example is provided as template in the ***ngcircon*** software module:

```
ngcircon/templates/xxircon
```

The ***xxircon*** template is installed in:

```
$INTROOT/templates/forNGC/
$VLTROOT/templates/forNGC/
```

The same applies to the general purpose control server. The only difference is that one needs to derive from the ***ngcdcsEVH*** class instead of the ***ngcirconEVH*** class and the ***ngcirconLib*** needs not to be defined in the Makefile. Also for this case an example is provided as template in the ***ngcdcs*** software module:

```
ngcdcs/templates/xxdcs
```

The ***xxdcs*** template is also installed in:

```
$INTROOT/templates/forNGC/
$VLTROOT/templates/forNGC/
```

26.2. State Switching Call-Backs

The following call-backs are provided when the server state changes (i.e. upon reception of an ***ONLINE***, ***STANDBY*** or ***OFF*** command):

```
ccsCOMPL_STAT OnlineCB1();
ccsCOMPL_STAT OnlineCB2();
ccsCOMPL_STAT StandbyCB1();
ccsCOMPL_STAT StandbyCB2();
ccsCOMPL_STAT OffCB1();
ccsCOMPL_STAT OffCB2();
```

The *xxxCB1()* functions are called before the state changes, the *xxxCB2()* functions are called after internal state switching.

26.3. SETUP/STATUS Call-Backs

The following call-backs are provided upon reception of a ***SETUP*** command:

```
ccsCOMPL_STAT SetupCB1(char **list, vltINT32 *size);
ccsCOMPL_STAT SetupCB2();
```

The *SetupCb1()* is executed before the internal setup is done. The setup *list* contains pairs of parameter names and values. The list always has to be examined within *SetupCb1()*. 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()*.

The following call-back is provided upon reception of a ***STATUS*** command:

```
int LookupCB(const char *name, char *value);
```

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. The function is also called whenever a parameter needs to be resolved from within the sequencer program.

26.4. Post-Processing Call-Back

The post-processing call-back is executed whenever a new data frame is received by the data acquisition thread of the control server:

```
int PostProcCB(void *buffer, ngcdcs_finfo_t *finfo, eccsERROR *error);
```

The ***ngcdcs_finfo_t*** structure *finfo* contains all information for the *buffer*:

```
int type;          - Unique frame type
char name[64];     - Unique frame name
int fcnt;          - Frame counter
double expFactor;  - Factor for EXPTIME
int bitPix;        - Bits per pixel as defined in the FITS-standard
int divisor;       - Pixel divisor
int bscale;        - Scaling factor to be applied
int bzero;         - Offset value to be added
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
int fcnt0;         - Initial frame counter
int acq;           - Acquisition module index
char utc[64];      - Time when frame was ready in the pre-processor
char where[512];   - Location where file has been stored (full path)
ngcdcsCUBE *cube;  - Data cube object to be used for storing to a cube
```

The ***ngcdcsCUBE*** class contains the following members:

```
FILE *fd;            - File descriptor
int naxis1;          - Dimension in x-direction
int naxis2;          - Dimension in y-direction
int naxis3;          - Number of images
int bitPix;          - Bits per pixel as defined in the FITS-standard
int divisor;         - Pixel divisor
int bscale;          - Scaling factor to be applied
int bzero;           - Offset value to be added
int sx;              - Lower left corner (x-direction)
int sy;              - Lower left corner (y-direction)
double expFactor;    - Factor for EXPTIME
char fileName[256];  - Actual filename (full path)
char frameName[64];  - Frame type name for all images in the cube
unsigned int cnt;    - Cube counter
int fcnt;            - Frame counter
int fcnt0;           - Initial frame counter
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 values:

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

When the call-back function returns ***ngcbSKIP*** then the string "*where*" of the ***ngcdcs_finfo_t*** structure can be set to the full path of the location where the data has been stored. When this is not an empty string then an appropriate file event will be generated by the control server. By default the string "*where*" is empty.

Furthermore file-events can be issued with the *SendFileEvent()* function.:

```
void SendFileEvent(const char *fileName);
```

This may be useful when the application produces several files out of a received data frame. When not being called from within the *PostProcCB()* the *SendFileEvent()* function has no further effect.

By default the *PostProcCB()* is only called when an exposure is ongoing. If the *PostProcCB()* needs to be called also when no exposure is active (e.g. secondary auto-guiding) then one has to enable this explicitly with

```
> setup DET.ACQi.TRANSFER T;
```

The parameter can also be set with the "*Transfer*"-button in the GUIs Acquisition-widget.

The post-processor runs asynchronously within its own tread. The default stack-size for the post-processor thread is 8M. The value can be changed by setting the public variable "***ppStackSize***" to the desired value (in bytes). The variable should be set in the constructor of the derived server class.

Caution:
Because of its concurrency with the event handler the post-processing call-back by default must not send or receive CCS-messages. This also affects online database access. When CCS-functions need to be called from within the *PostProcCb()* then a locking mechanism needs to be employed. This can be achieved with the function *UseEvhLock()* which should be called exactly once e.g. in the constructor of the derived class. Furthermore all CCS-compliant functions executed inside the call-back must use the *error-stack*, which is passed as parameter to the callback.

Caution:
Generally the implementation of a post-processing call-back always includes some pitfalls such as the synchronization with external tasks, shared memory limitations, transfer of partial images or the parameterization of the post-processing algorithm. Also the execution of the post-processing algorithm must be faster than the data are coming in (otherwise frames will be skipped).

A much simpler approach is always to operate on the FITS-file(s) produced by the server during the exposure. In many cases the file-access overhead is not the limiting factor. Fast storage devices such as solid state drives will make the overhead become negligible in future. Where real-time constraints become tighter an implementation of the algorithm in the acquisition process on the NGC-LLCU will be mandatory anyway. So there is only a small gap left for applications to really benefit from the *PostProcCB()*.

26.5. FITS-Header Completion Call-Back

The FITS-header completion call-back is executed once just before a FITS-file (data cube or image extension format) is closed:

```
int CompleteFitsCB(ngcbFITS *hdr, eccsERROR *error);
```

The application may then add its own keywords to the primary header.

Example:

```
int ngcdcsEVH::CompleteFitsCB(ngcbFITS *hdr, eccsERROR *error)
{
  int ret = ngcbSUCCESS;
  int myValue = 1;

  if (hdr->Add("DET.MYPARAM", myValue, "My Comment") != ngcbSUCCESS)
    {
    error->ErrAdd(ngcdcsMOD_ID, ngcdcsERR_EXP_PROC, __FILE_LINE__,
         hdr->ErrMsg());
    return (ngcbFAILURE);
    }

  return (ret);
}
```

Keywords may be added as *integer*, *float* or *double* types or directly as a string (see man-page of the ***ngcbFITS*** class).

27. Some Useful Tools

27.1. ngcguiModVersions

Syntax:

```
ngcguiModVersions [host]
```

The command prints out the version and date of all NGC software modules installed on *host*. If *host* is not specified the local host is taken by default.

Additionally the actually installed version of all modules listed in section 2.1 can be retrieved with a "<*module-name*>*Version*" command (e.g. "*ngcppVersion*").

27.2. ngcguiCheckEnv

Syntax:

```
ngcguiCheckEnv
```

The command returns the exit status (0) when the CCS environment is running. In case the registration with the CCS environment has failed, a message indicating the reason for failure is written to the *stderr* stream and the exit status (1) is returned. The ***$RTAPENV*** environment variable is used to determine under which CCS-environment the application needs to be registered.

27.3. ngcdcsFaq

Syntax:

```
ngcdcsFaq
```

Frequently-asked-questions list browsing tool (see section 28.1).

27.4. ngcbCatFits

Syntax:

```
ngcbCatFits [hdu] <file name>
```

The command lists the FITS header units of a *.fits* file. If *hdu* is zero (or not defined) all header units are displayed.

27.5. ngcbCube2Ext

Syntax:

```
ngcbCube2Ext <file name> [extension name]
```

The command is used to move a data-cube to a FITS-extension. When no extension name is given then the frame type name as read from the ***DET.FRAM.TYPE*** keyword will be used as ***EXTNAME***.

27.6. ngcbSplitCube

Syntax:

```
ngcbSplitCube <source-file> [target-file]
```

The command is used to split data-cube to into FITS-extensions per chip. When no target file is supplied as second argument then the source file is replaced. In case an error occurs the source file is not touched bit it may happen that, if the file renaming fails, the files may remain in:

```
<source-file>.orig - Original source file
<source-file>.tmp  - Temporary target file
```

27.7. ngcbCube2Rtd

Syntax:

```
ngcbCube2Rtd <file name> [camera name] [-nodisp] [-nowish]
```

The command displays a data-cube in the RTD-application. By default the RTD-application is launched by this tool. When the *-nodisp* option is present then no RTD-application is launched and only the image events are sent to the camera. When no camera name is given then the camera name is read from the environment variable ***$RTD_CAMERA***. When the environment variable is not defined then the name "***RTD_NGC***" will be used by default.

Unless the *-nowish* option is set the tool will raise a control widget. Otherwise the control is done through *stdin*:

```
"+"   (next slice),
"-"   (previous slice),
"."   (refresh),
"<n>" (set slice number to <n>),
"x"   (exit)
```

27.8. ngcbCmd

Syntax:

```
ngcbCmd [-host <host[:port]>] <command string>
```

Low-level command interface for the NGC control server. The communication is done through a TCP/IP socket interface. Both control server and ***ngcbCmd*** tool read their default communication port from the environment variable ***$NGC_PORT***. Unless a specific port number is assigned it must be ensured that this environment variable is the same for both the control server and the shell where ***ngcbCmd*** is launched.

If the control server is started with the *-inst <label>* option (i.e. if the startup configuration parameter ***DET.CON.INSTANCE*** is not an empty string) then the server assumes that it will not be the only instance running and it will not open the default port. In this case an explicit port number must be assigned to open the communication port. This can be done

a) in the startup configuration:

```
DET.CON.SRVPORT <port-number>;
```

b) via the startup tool command line

```
ngcdcsStartServer ... -port <port-number>
```

c) via the server command line (when not using the startup tool):

```
ngcdcsSrv ... -port <port-number>
ngcdcsEvh ... -port <port-number>
ngcircon  ... -port <port-number>
```

The ***ngcbCmd*** tool must then also be told to which server it should send the command. This is done with the *-host* option. When the *-host* option is not given at all then ***$HOST:$NGC_PORT*** is applied. When just the port number is not given then ***ngcbCmd*** sends the command to the specified host but reads the port from the ***$NGC_PORT*** environment variable.

The commands *help*, *help <command>* and *help <all>* provide information about all implemented commands. The ***ngcbCmd*** command is foreseen for engineering and detector development purposes only.

27.9. ngcbParamInfo

Syntax:

```
ngcbParamInfo [-w <n>] <name> [dictionary-1] [dictionary-2]
[...]
```

Lookup parameter *name* in the given dictionaries and print the parameter description to *stdout*. The dictionary ***ESO-VLT-DIC.NGCDCS*** is always included and needs not to be specified. Unless an absolute path is given the dictionary names can be specified without the leading "*ESO-VLT-DIC.*" string.

The "*-w*" option defines a field width for formatted output (word-wrapping). When the given width is zero then no formatting is applied. The default field width is 60.

The environment variables ***$INS_ROOT*** and ***$INS_USER*** are used to build the search path for the dictionaries unless an absolute path is given. If the ***$INS_USER*** environment variable is not set, then the default value *SYSTEM* is assumed.

The search path is set to: *$INS_ROOT/$INS_USER/Dictionary*

27.10. ngcppSimple16/32

Syntax:

```
ngcppSimple16 [-nclient <n>] [-nx <pix>] [-ny <pix>]
ngcppSimple32 [-nclient <n>] [-nx <pix>] [-ny <pix>]
```

Simple data acquisition processes which can be used in stand-alone mode to acquire NGC raw-data (16-bit or 32-bit). The default value for the "*-nx/-ny*" options is *256x256*.

When the "*-nclient*" option is present and *n* is larger than zero then the data can be viewed in the RTD (start RTD with the *ngcrtd* command – see section 14.5).

The "*-nclient/-nx/-ny*" options must not be set when the process is launched via the control server.

When the process is started manually it uses by default the ***$NGCPP_DATA*** environment variable for the data-port number. The value can be overwritten with the *-dataport <number>* command line option.

27.11. ngcppTemplate

Syntax:

```
ngcppTemplate [-im sim] [-nclient <n>] [-nx <pix>] [-ny <pix>]
```

Data acquisition processes which can be used in simulation mode to test various features. The default value for the "*-nx/-ny*" options is *256x256*. The process produces simulated noise for DIT-frame and INT-frame and a ramp for the special frame type "*MYFRAME*"

When the "*-nclient*" option is present and *n* is larger than zero then the data can be viewed in the RTD (start RTD with the *ngcrtd* command – see section 14.5).

The "*-nclient/-nx/-ny/-im*" options must not be set when the process is launched via the control server.

When the process is started manually it uses by default the ***$NGCPP_DATA*** environment variable for the data-port number. The value can be overwritten with the *-dataport* <*number*> command line option.

27.12. ngcppAdc

Syntax:

```
ngcppAdc [-nadc <n>] [-nclient <n>] [-nx <pix>] [-ny <pix>]
```

Simple data acquisition processes which can be used in stand-alone mode to acquire NGC raw-data (16-bit). The data is sorted into a number of stripes as given with the "*-nadc*" option (default is 4). The y-dimension as specified by the "*-ny*" option must be a multiple of the number of ADCs. The default value for the "*-nx/-ny*" options is *256x256*.

When the "*-nclient*" option is present and *n* is larger than zero then the data can be viewed in the RTD (start RTD with the *ngcrtd* command – see section 14.5).

The "*-nclient/-nx/-ny*" options must not be set when the process is launched via the control server.

When the process is started manually it uses by default the ***$NGCPP_DATA*** environment variable for the data-port number. The value can be overwritten with the *-dataport* <*number*> command line option.

A 32-bit version of this tool is also available (*ngcppAdc32*).

27.13. ngcppDart

Syntax:

```
ngcppDart [options]
```

This is a simple data reception task which can be used to test the data transfer. The following options are supported:

```
-verbose <level>    - verbose level (default: 0)
-o <name>           - write data to file (.bin or .dat)
                      <name>.bin = binary, <name>.dat = ASCII
                      (default: <name>.bin)
-nx <pixels>        - set maximum x-dimension (default: 1024)
-ny <pixels>        - set maximum y-dimension (default: 1024)
-host <host name>   - set name/ip-address of data server
                      (default: $NGCPP_HOST)
-port <portNum>     - set port number (default: $NGCPP_DATA)
-type <type>          - set frame type number (default: -1 =
"all")
-video              - switch to video transfer mode
                      (default: science transfer)
-win <sx sy nx ny> - set window (default: full frame = [1 1 0
0])
```

The (manually started) acquisition process requires the "-nclient" option (with the number of clients being larger than zero) in order to get the data with this tool.

27.14. ngcbTestDma

Syntax:

```
ngcbTestDma [options]
```

DMA data transfer test routine. The following options are supported:

```
-dev <name>     - DMA device name (Default: /dev/ngc0_dma)
-size <size>    - buffer size in bytes (Default: 16384)
-num <n>        - number of buffers to read (Default: 1)
-tmo <seconds> - timeout (seconds)(Default: -1)
-dump <num>     - dump <num> pixel values to stdout
```

The process opens the device and reads *n* times a buffer of the given *size*. A timeout value can be defined in seconds ("-1" indicates an infinite timeout).

28. Troubleshooting

28.1. Frequently Asked Questions

The *NGCIRSW* maintains a list of *Frequently Asked Questions* (***FAQ***). The list can be displayed with the "***ngcdcsFaq***" browsing tool once the software is installed. It is also available through the "*Help → Extended Help*" menu button in the graphical user interfaces "***ngcgui***" and "***ngcguiHw***". The list always refers to the software version actually installed.

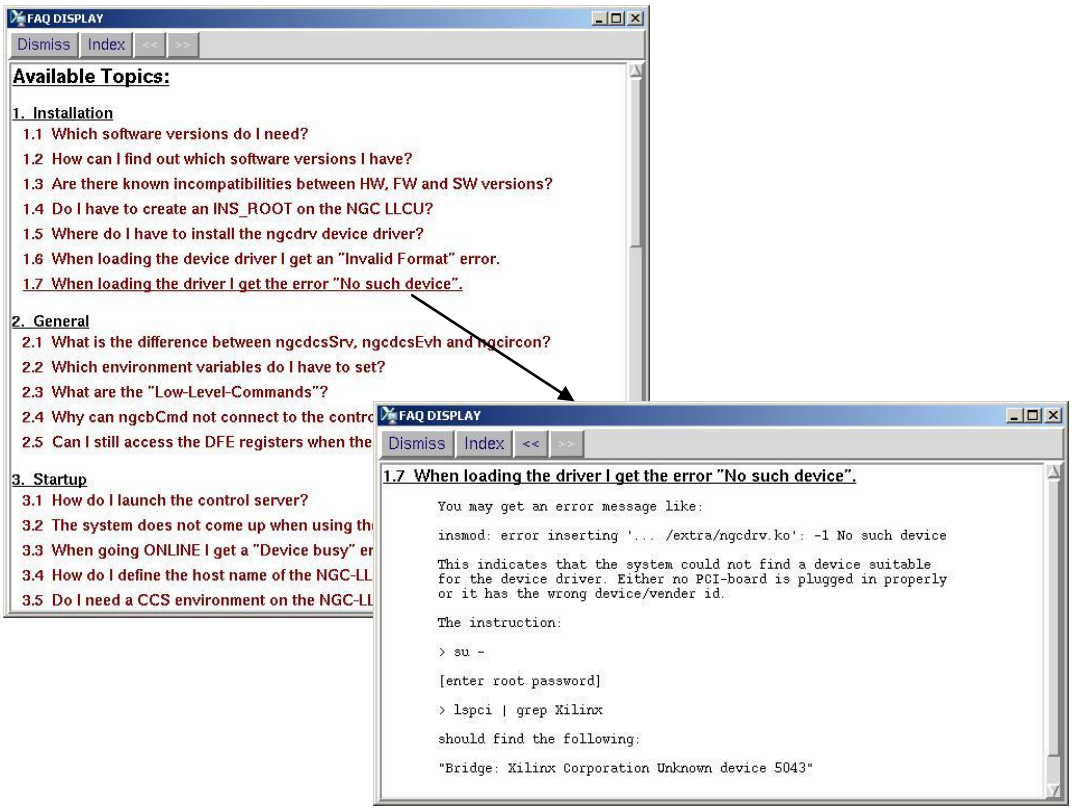

Figure 39 FAQ-List Display

A printable ASCII-file is provided in the ***ngcdcs*** software module:

```
ngcdcs/config/ngcdcs.faq
```

The file is installed at the following location:

```
$INTROOT/config/ngcdcs.faq
$VLTROOT/config/ngcdcs.faq
```

28.2. Man Pages

The *NGCIRSW* provides man-pages for all relevant processes, classes and library functions. Main entry points are:

- `man ngcdcsStartServer`
- `man ngcdcsStopServer`
- `man ngcdcsStartGui`
- `man ngcircon`
- `man ngcdcsEvh`
- `man ngcdcsACQ_DATA_CLASS`
- `man ngcdcsMerge`
- `man ngcdcsEXPDRV`
- `man ngcdcsAOCAL`
- `man ngcgui`
- `man ngcrtd`
- `man ngcrtdDtt`
- `man ngcrtdCtrl`
- `man ngcdrv`
- `man ngcbDrvCom`
- `man ngcbDrvDma`

Also the man-pages always refer to the software version actually installed.

28.3. Problem Reporting

Problems and change requests can be reported through the ESO JIRA ticketing system.

http://jira.eso.org

When not having access to this system you may send an e-mail to:

ngc@eso.org

Please always mention the software module versions you are actually using. The versions can be retrieved with the command

"`ngcguiModVersions`"

(see section 27.1) or via the "*Help → About*" menu button in the graphical user interfaces "***ngcgui***" and "***ngcguiHw***".