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

INSTRUMENTATION DIVISION

New General detector Controller

Infrared Detector Control Software – User Manual

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

Document Issue: **<2>**

Date of Issue: **<08/10/2008>**

	Name	Date	Signature
Prepared by :	Jörg Stegmeier	8.10.08	[illegible]
Approved by :	Dietrich Baade, Gert Finger	27.10.2008 27.10.2008	Dietrich Baade Finger Gert
Released by :	Mark Casali	27/10/08	[illegible]

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. Added new command line options. Added new section for NGCIRSW operational states. Aligned order with actual CDT. Updated
0.3	26-05-2007	9.10 16, 17 16.4	Burst-Mode Added configuration section. Added read-out mode section
0.4	30-05-2007	19	Server extensions
0.5	06-06-2007	11 19.4	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, 6.3, 6.4 9.7 9.6 18 20	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. Added frame-type parameters. I^{2}C interface added. Shutter interface added.
2.0	08-10-2008	None	New issue number.

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*).

Its purpose is to provide people, who intend to use the NGC electronics for infrared instruments, with all the necessary information to install from scratch the *NGCIRSW*, interact programmatically with the *NGCIRSW*, or operate an infrared 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 [RD78]. Basically the NGC electronics [AD8] 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 and IRACE, different SW-architectures have been chosen, which are described in detail in the NGC SW design documents [AD9], [AD10] and [AD11]. 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 13) 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 basically the demands on process concurrency are very different in both cases.

In order to optimize the communality 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 15) 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*. 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

Applicable documents used in the NGC project are listed in the document VLT-LIS-ESO-13660-3906 "NGC Project Documentation".

1.4. Reference Documents

Reference documents used in the NGC project are listed in the document VLT-LIS-ESO-13660-3906 "NGC Project Documentation".

1.5. Abbreviations and Acronyms

Abbreviations and acronyms used in the NGC project are listed in [AD64].

1.6. Glossary

All the relevant concepts used within the NGC project are listed in [AD63].

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 [AD27].

1.9. Problem Reporting/Change Request

The form described in [AD72] shall be used.

2. Installation

2.1. Software Modules

All software modules are under CMM configuration control.

- ***dicNGC*** - A dictionary, which is common to both infrared and optical systems [AD9, RD10].
- ***ngcdrv*** - The device driver for the PCI-Bus back-end card [AD9].
- ***ngcb*** - The NGC base software module containing the driver interface library (***ngcbDrv***) for communication and DMA, some basic i/o tools, a portable threads- and priority-control implementation and the C++ base classes for general system access. This module also provides a simulation mechanism for the NGC controller. See [AD9] for details.
- ***ngcpp*** - The DMA data-acquisition and pre-/processing module [AD9].
- ***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 [AD9]. 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 [AD11].
- ***ngcircon*** - The NGC system control module for infrared applications [AD11]. This includes the infrared control server instance and all additional infrared specific device classes (chopper, special ADC or CLDC hardware releases, etc.).
- ***ngciracq*** - The default acquisition processes for NGC infrared detector systems.
- ***ngcrtd*** - The NGC RTD application.
- ***ngcarch*** - Installation scripts for the overall *NGCIRSW* package (see section 2.2).
- ***ngcBUILD*** - Installation module for ***pkgin*** (latest versions).
- ***ngcins*** - Installation module for ***pkgin*** (fixed versions).

2.2. Installation Scripts

Installation scripts for the entire *NGCIRSW* package are provided in the ***ngcarch*** software module. The ***ngcarch*** module 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 2.4.1 for the device driver installation. The procedure to install the software consists of the following steps:

1. Retrieve the ***ngcarch*** module from the archive and prepare installation:

```
mkdir <NGCROOT>
cd <NGCROOT>
cmmCopy ngcarch
```

2. Retrieve from the archive and install all the other modules:

```
cd ngcarch/src              or    cd ngcarch/src
make prepare_installation         make all install
cd ../../INSTALL
./buildAll
```

3. As an alternative to point 2, if the modules have already been retrieved from the archive and only the installation is needed:

```
cd <NGCROOT>/INSTALL        or    cd ngcarch/src
./buildNGC                        make install
```

4. As an alternative to point 2 or 3, if only the modules should be retrieved from the archive without doing the installation:

```
cd ngcarch/src              or    cd ngcarch/src
make prepare_installation         make all
cd ../../INSTALL
./buildFromArchive
```

This procedure uses fixed versions of all modules. In order to install the latest versions one has to replace:

"*buildFromArchive*"	→ "*buildLatestArchive*",
"*buildAll*"	→ "*buildLatest*",
"*make all*"	→ "*make update*"
"*make all install*"	→ "*make update install*".

2.3. Installation with pkgin

To patch the NGC SW (both optical and infrared) for a certain instrument insert in the instrument <xx>insINSTALL.cfg the following lines:

```
INSTALL.MODULE<N>.NAME "ngcins"
INSTALL.MODULE<N>.VERSION "<version>"
INSTALL.MODULE<N>.OPTIONS "LIBRARY"
INSTALL.MODULE<N>.SUBPKG "VLTSW_new"
```

See [RD31] for details concerning VLTSW package installation.

2.4. Manual Installation

2.4.1. Driver Installation

Use the following instructions to retrieve and install the NGC device driver module on the NGC-LLCU (installation is not required on the IWS):

- *cmmCopy ngcdrv*
- *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*".

2.4.2. Control Software Installation

Use the following instructions to install the control software on both the NGC-LLCU and the IWS:

a) Retrieve Modules:

- cmmCopy ngcdrv
- cmmCopy dicNGC
- cmmCopy ngcb
- cmmCopy ngcpp
- cmmCopy ngcdcs
- cmmCopy ngcgui
- cmmCopy ngciracq
- cmmCopy ngcircon
- cmmCopy ngcrtd

b) Compilation and Installation:

- cd dicNGC/src
- make all install

- cd ngcdrv/src
- make man all install

- cd ../../ngcb/src
- make man all install
- cd ../test
- make man all install

- cd ../../ngcpp/src
- make man all install
- cd ../test
- make man all install

- cd ../../ngcdcs/src
- make man all install

- cd ../../ngcgui/src
- make man all install

- cd ../../ngciracq/src
- make man all install

- cd ../../ngcircon/src
- make man all install

- cd ../../ngcrtd/src
- make man all install

Caution:
The modules should be compiled in that order. When patching a single module the dependencies have to be considered (i.e. recompile all higher level modules).

3. Startup/Shutdown Procedure

3.1. System Configuration

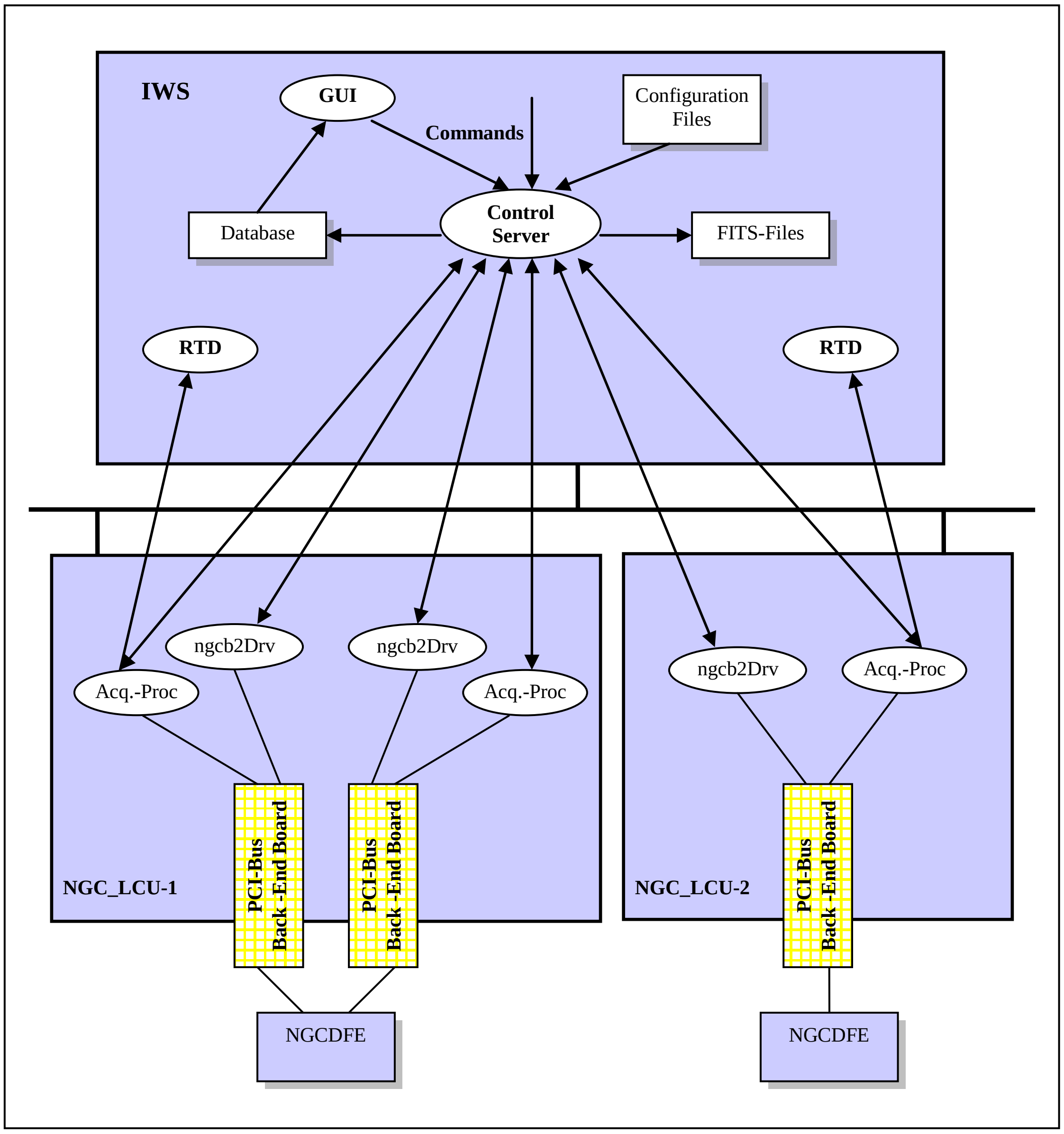

Figure 1 System Architecture

The NGC infrared detector control software (*NGCIRSW*) is running partly on the instrument workstation (IWS) and 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-Bus 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-Bus 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 [AD9]. 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.

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 and detector configuration. 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 separated in order to avoid to unnecessarily duplicate the information.

The configuration files are kept in separate instrument specific configuration modules ("***xxdcfg***"), which are under CMM-control. 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 15) is already provided in the ***ngcdcs*** SW module. The following sections are also applicable to this basic server by replacing the process name 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***' will be used. The instance label (if given) will always be added 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 this option is not present, then the file-name "*NGCIRSW/ngc.cfg*" is used by default (this only applies to the ***ngciron*** server but not to the general purpose control server). When the file-name is "none", a built-in default configuration is applied (this is only valid for test-purposes).
`-mode <mode>`	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).
`-online`	Go automatically to ***ONLINE***-state after startup.
`-start`	Automatically start sequencer(s) when going to ***ONLINE***-state.
`-poll`	Enable sub-system status polling. This will periodically read the controller electronics status.

`-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).
`-help or –usage`	Show available command line options.

Table 1 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 14):

```
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 1).

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 HW 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***", [RD75]). 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***' will be used. The instance label (if given) will always be added 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 this keyword is not present, the file-name "*NGCIRSW/ngc.cfg*" is used by default. When the file-name is "none", a built-in default configuration is applied (this is only valid for test-purposes).
`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.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 whit-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. 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 (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. 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) in case the keyword is not present.

Table 2 Startup Configuration Keywords

Each configuration module ("***xxdcfg***") contains an "*xxdcfgCONFIG.cfg*" file, which assigns a name and some access-right attributes to each of those 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 configuration:

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

This will load the given startup configuration, start the server process and wait with a default timeout until the server is actually running (i.e. the server is responding to ***PING*** commands). Configuration-set names are typically "XXDCS", where "XX" corresponds to the two-letter instrument code.

The command line options as given in Table 1 can still be passed through to the server to override the corresponding values in the configuration set keywords (Table 2). If no configuration set is given, only the command line options are used. Some special options (which are not passed through) can be specified to tune the behavior of these startup scripts:

```
-restart        -   restart server/gui if still running
-nowait         -   do not wait until server/gui is up
-timeout <ms>   -   timeout for ping (in milliseconds)
-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 through a script 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.

3.2.3. Examples

<u>Startup Configuration (*xxdcfgCONFIG.cfg*):</u>

```
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";

# 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 [AD28]):

- **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 number) 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.

Figure 2 illustrates the *NGCIRSW* operational states and the commands to switch between them.

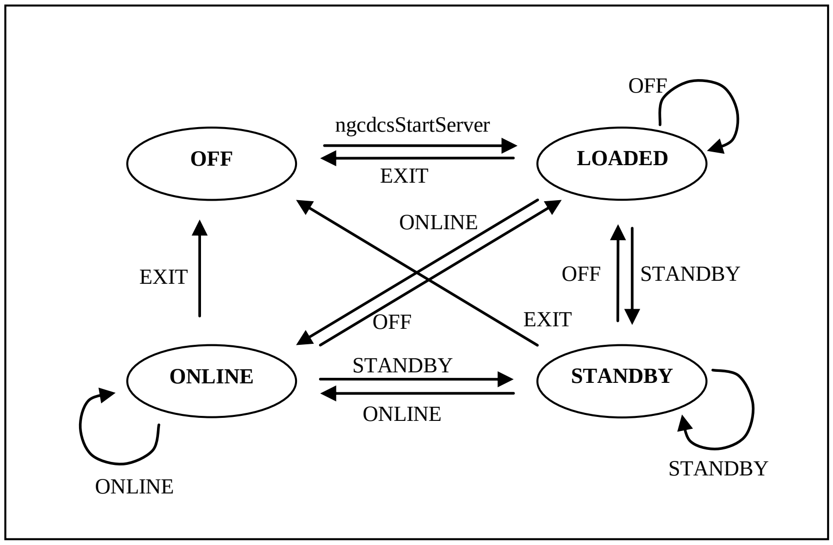

Figure 2 Operational States and State Transitions

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 ignored.		
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.
	-zero	Logical	Set all voltage channels to zero volts.
	-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.
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 ignored.		
EXIT	Exit server. *NGCIRSW* will go to ***OFF*** state.		
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).
	-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.		
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`	Issue a NGC command (ASCII string). This is a low-level interface to all NGC controller electronics functions.		
`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 $INS_ROOT.
	`-tmo`	Integer	Two values giving the timeout (in seconds) for the wait instruction (1st) and a polling interval (2nd). The default polling interval is 100 ms in case only the first parameter is set.
	`-wait`	String	Wait for the specified sequencer program event (“trigger” | “break” | “end”).
	`-stop`	Logical	Stop sequencer and all associated acquisition processes.
	`-start`	Logical	Start sequencer and all associated acquisition processes.
	`-pause`	Logical	Pause sequencer program execution.
	`-cont`	Logical	Resume sequencer program execution.
	`-trigger`	Logical	Send software trigger.
`SETUP`	**`-expoId`**	Integer	Ignored.
	`-file`	String	Load setup from a file. Unless an absolute path is given, the setup file is searched in the directory `$INS_ROOT/$INS_USER/COMMON/SETUPFILES`
	`-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 a list of all dynamically defined parameters is returned (i.e. those which are actually in use for the current read-out mode).
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 3 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 8. 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 10). 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
#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 system   (NGC system parameters)
                              |--o exposure (exposure parameters)
                              |--o mode     (read-out mode parameters)
                              |--o guiding  (guiding parameters)
                              |--o chopper  (chopper interface)
                              |--o seq_<i>  (sequencer parameters)
                              |--o cldc_<i> (CLDC parameters)
                              |--o adc_<i>  (ADC module parameters)
                              |--o acq_<i>  (acquisition module parameters)
```

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

Database classes are provided for all sub-branches as described in the following sections. The classes are part of the ***ngcdcs*** SW module (TBD: "*guiding*" and "*chopper*") and are also used by the general purpose control server (see section 15 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 system operation mode (NORMAL, SIM-HW, SIM-LCU).
stateName	BYTES32	System status (string).
state	INT32	System status.
subStateName	BYTES32	System sub-state (string).
subState	INT32	System sub-state.
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.
fitsHdrSize	INT32	Number of FITS header block to reserve for merging.
polling	INT32	Status polling flag.
currentAction	BYTE256	Action log.
param	Table	128 dynamic parameters (BYTES32 name, BYTES32 value).

Table 4 System Database Class (ngcdcsSYSTEM.class)

6.2. 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 (string: “running”, “idle”, “waiting”, “failure”).
status	INT32	Sequencer status.
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).
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).

Table 5 Sequencer-Module Database Class (ngcdcsSEQ.class)

6.3. CLDC Module Database Class

Attribute	Type	Description		
Name	BYTES64	CLDC module name.		
cfgFile	BYTES256	Voltage configuration file.		
statusName	BYTES32	CLDC status (string, “enabled” or “disabled”).		
status	INT32	CLDC status.		
clkMon1	INT32	Clock line on clock monitor 1.		
clkMon2	INT32	Clock line on clock monitor 2.		
clkSwitch	INT32	Clock-switch register value.		
preampOffset	DOUBLE	Preamplifier offset.		
diodeBias	DOUBLE	Diode bias.		
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 (20 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 6 CLDC-Module Database Class (ngcdcsCLDC.class)

6.4. 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.
packetCnt	INT32	Packet routing length (number of packets from down-link).
Convert1	INT32	Enable/disable conversion strobe-1 (0|1).
convert2	INT32	Enable/disable conversion strobe-2 (0|1).
opMode	INT32	Operational mode.
simMode	INT32	Simulation mode.
Monitor1	INT32	Monitor 1.
monitor2	INT32	Monitor 2.
offset	DOUBLE	ADC offset.
Clamp	INT32	Clamp
Filter	INT32	On-board filter (0 = 0.5 us, 1= 5 us)

Table 7 ADC-Module Database Class (ngcdcsADC.class)

6.5. Acquisition Module Database Class

Attribute	**Type**	**Description**		
name	BYTES64	Acquisition module name.		
statusName	BYTES32	Acquisition module status (string: “idle”, “ready”, “running”, “failure”).		
Status	INT32	Acquisition module status.		
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	Enable/disable sustained data transfer (0|1).		
guiding	INT32	Enable/disable secondary auto-guiding (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 8 Acquisition-Module Database Class (ngcdcsACQ.class)

6.6. Exposure Database Class

Attribute	Type	Description
expStatus	UINT32	Exposure status (from *inscEXPOSURE.class*).
expStatusName	BYTES32	Exposure status name.
dataPath	BYTES256	Path to the directory where detector data is stored.
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. Starts from exposure time estimation value and decrements every second after exposure start until the exposure has completed.
extFits	INT32	Generate extended FITS header.

Table 9 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 10 Read-Mode Database Class (ngcdcsMODE.class)

6.8. Guiding Mode Database Class

Attribute	Type	Description
offset	Vector (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 11 Guiding-Mode Database Class (ngcdcs2AG.class)

6.9. 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).

Table 12 Chopper Database Class (ngcdcsCHOPPER.class)

7. Setup Command

This section shows a selection of the most important keywords (to be completed).

Keyword	Type	Description
`DET.FRAM.NAMING`	`String`	Defines the frame naming scheme ("*request*", "*sequence*" or "*auto*") as described in section 9.4.
`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*") 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.
`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.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.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.
`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 like detector saturation during re-start.
`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.

Keyword	Type	Description
DET.ACQi.BURST.NUM **DET.ACQi.BURST.SKIP**	**Integer**	Parameters for the burst-mode as described in section 9.10.
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.
DET.CLDCi.FILE	**String**	Load a new voltage file.
DET.CLDCi.CLKOFF	**Double**	Clock voltages offset on CLDC module.
DET.CDLCi.DCOFF	**Double**	Bias voltages offset on CLDC module.
DET.CLDCi.CLKHIi	**Double**	Set voltage for the high level of the given clock. The range will be checked before the new value is applied.
DET.CLDCi.CLKLOi	**Double**	Set voltage for the low level of the given clock. The range will be checked before the new value is applied.
DET.CLDCi.Dci	**Double**	Set voltage for the given bias channel. The range will be checked before the new value is applied.
DET.CLDCi.PREAMP	**Double**	Preamplifier offset voltage.
DET.CLDCi.DIODE	**Double**	Diode bias voltage.
DET.CLDCi.CKSWITCH	**Integer**	Clock-switch register value.
DET.CLDCi.MONi	**Integer**	Selects the clock to be monitored on clock monitor 1 or 2.
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 2 is not supported by the current hardware revisions).
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).

Table 13 Setup Commands

8. Status Command

This section shows a selection of the most important keywords (to be completed).All of the ***SETUP***- keywords described in section 7 can be retrieved via the ***STATUS*** command.

Keyword	**Type**	**Description**
`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.FRAME.AVAIL`	**`String`**	Returns a list with all frame-types currently defined (see section 9.5).
`DET.CHOP.FREQ`	**`Double`**	Returns a maximum value for the chopping frequency to be used with currently loaded sequencer program.
`DET.CON.OPMODE`	**`String`**	Returns the current operational mode of the detector front-end system (NORMAL, HW-SIM or LCU-SIM).

Table 14 Status Commands

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 anyway 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 current exposure status value is stored in the database attribute '***<alias>ngcircon:exposure.expStatus***'. It can take 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 mode the counter-reset is disabled and it is up to the initiator of the exposure start command to ensure, that there was no change in the field of view since the last integration start. A timed exposure start can be done using the command

```
START –at <hh.mm.ss[.uuuu]>
```

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

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***").

<u>Changes with respect to IRACE:</u>

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 this, the frames may be stored into data-cubes (***DET.FRAM.FORMAT = "cube"***). One cube would be created per frame type. This is especially needed for storing data in burst mode, where usually only very small windows are read out. To store several thousands of those small windows in binary image extensions or even single files would imply an enormous overhead.

The standard file format 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 will normally be available all at the same time. When storing to different files (i.e. one FITS-file per acquisition process containing all frames delivered by this process), all transfer can be done in parallel and the transfer processes need not to wait for each other before saving data to disk (***DET.FRAM.MULTFILE = "T"***). In the right configuration this would improve the transfer performance considerably. Nevertheless, when transfer performance is not the limiting factor, the storage mechanism is configurable to have only one binary image extension file per exposure (***DET.FRAM.MULTFILE = "F"***). A default value for this can be 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 **<u>undefined</u>**! In any case each image extension contains a unique identifier in the extension header.

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

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

<u>Changes with respect to IRACE:</u>

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).

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***":

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

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

6. ***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
   ```

7. ***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).

<u>Changes with respect to IRACE:</u>

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 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>] [-win sx sy nx ny]***”

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

The available frames are stored as a table (fields: *name*, *generate*, *store*, *breakCond*, *sx, sy, nx, ny*) for each acquisition module in the data base attribute ‘***<alias>ngcircon:acq_<i>.frame***’.

The frame list can also be received with the command “***STATUS -function DET.READ.FRAMES***”. 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 break counter for the exposure termination condition. [sx sy nx ny] gives the software window to be applied for this frame.

<u>Changes with respect to IRACE:</u>

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";     # acquisition process list (STDEV on process 1)

DET.FRAM3.NAME "STDEV"; # frame name
DET.FRAM3.ACQ  "2,3";   # acquisition process list (STDEV on process 2, 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.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.FRAME1.NAME = "DIT"
DET.FRAME2.NAME = "INT"
DET.FRAME3.NAME = "STDEV"
DET.FRAME4.NAME = "INTERM-INT"
DET.FRAME5.NAME = "HCYCLE1"
DET.FRAME6.NAME = "HCYCLE2"
DET.FRAME7.NAME = "SAMPLE"
DET.FRAME8.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.FRAMEi.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 9.6) 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.

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 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 TRIGGER= F / Trigger mode active
HIERARCH ESO DET SEQ1 TRIGMODE= 0 / Trigger mode (0=direct,1=shutter)
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
```

<u>Changes with respect to IRACE:</u>

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.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 3) with an appropriate sorting map. It is possible to split them up again into image extensions when storing the data to disk and use the full image only for the real-time display. The ***DET.ACQ.SPLITX/Y*** keywords define how the chips are mapped in the full image in x- and y-direction.

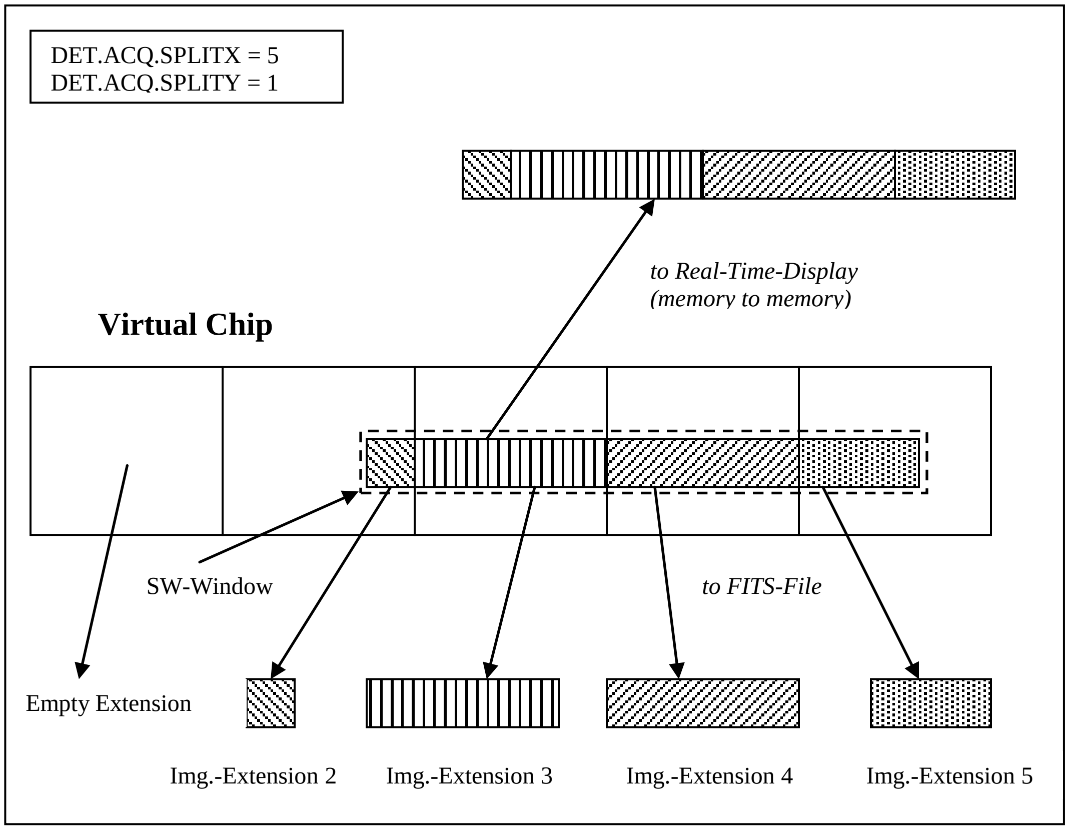

Figure 3 Virtual Chip and Overlapping Windows

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 setup parameters ***DET.WIN.STRX***, ***DET.WIN.STRY***, ***DET.WIN.NX***, ***and DET.WIN.NY*** define the format and position of the data-frame within the chip. ***DET.WIN.STRX/Y*** always refers to the lower left corner. The sequencer modules will derive the default read-out window (***DET.SEQi.WIN.STRX/ Y***, ***DET.SEQi.WIN.NX/Y***) from these parameters (TBD). For detector mosaics (***DET.CHIPS*** > 1, see section 9.6) 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. A zero value for the dimension keywords (***xxxx.WIN.NX/Y***) generally indicates a "full-frame" read-out.

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

Software-windows (i.e. windows which are applied after acquisition has been done but before data is transferred) can be set individually for each frame type as described in section 9.5. Chip overlapping windows for virtual chips are taken into account as shown in Figure 3.

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. Setup keywords had to be changed accordingly to support the new features.

9.10. Burst Mode

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. Two kinds of "burst modes" are used to cover these two cases.

9.10.1. Raw Data Mode

The raw data mode is activated by sending the setup command

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

If ***num*** is greater than zero, it indicates the number of [***DET.ACQi.WIN.NX, DET.ACQi.WIN.NY***]-dimension sample-frames to be stored in the burst buffer. If an exposure is started, both sequencer and acquisition are restarted (the ***DET.SEQi.CONT*** flag is ignored in that case) and the buffer is filled until ***num*** sample-frames (16 bit short integer) are stored. At the end of the exposure an INT-frame (containing only dummy data) is transferred. The transfer of the sample-frames starts immediately and runs in parallel to the data recording. 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 default burst process (*ngcppBurst*) applying the described mechanism may be changed by specifying a user-defined burst-process (***DET.ACQi.BURST.PROC***).

9.10.2. Internal Burst Mode

The internal burst is activated by sending the setup command

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

The negative value indicates that an internal burst-buffer should be applied. In this case the DMA is enlarged by a factor of ***num*** and soft-interrupts are created for each sub-division. The data processing is not affected in this case, but depending on the actual processing algorithms the performance may be slowed down. Also in this case a value of zero for ***DET.ACQi.BURST.NUM*** deactivates the burst-mode.

<u>Changes with respect to IRACE:</u>

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. Synchronization

Synchronization points can be inserted at any place in any clock pattern executed by the sequencer program (i.e. set the "*wait-for-trigger*" bit in the particular state). When reaching such a point, the pattern execution is suspended until the arrival of an external trigger signal (see [AD9] and [AD8] for signal timing and accuracy). The "*wait-for-trigger-state*" is only activated when the keyword ***DET.SEQi.TRIGGER*** is set to "***T***". Otherwise the "*wait-for-trigger-state*" is just passed through. Via the external trigger input it is possible to synchronize exposures on multiple *NGCIRSW* instances. The external trigger signal is also used to synchronize detector read-outs with external devices (e.g. chopper, see section 20). Using the VLT-TIM for generating the trigger pulse(s), synchronization at absolute times is possible. Some signal lines are available to in turn trigger external devices (e.g. tell another device, that a read-out has finished).

The trigger can also be sent via command:

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

Here <***n***> is sequencer instance number. If not specified the first module (***n*** = ***1***) is assumed.

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*** in the detector configuration file). 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, 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 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.

11. Error Definitions

The CCS error mechanism [RD32] 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***, ***ngcirconERR_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 15 and Table 16 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 behaviour.
`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 15 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.

Error	**Severity**	**Description**
`ngcdcsERR_WARNING`	Warning	A warning which has only very limited effect on the further system behaviour.
`ngcdcsERR_FATAL`	Fatal	An error which cannot be recovered in any case.
`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.
`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, then the ring-buffer may overrun. Go ONLINE again and either read-out slower or add sufficient delays between the read-outs.

Error	**Severity**	**Description**
`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.
`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 16 Error Definitions (*ngcdcs* SW module)

Changes with respect to IRACE:

The error definitions are not backwards compatible with IRACE.

12. 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 [AD27], [AD32]). 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.

13. Real-Time Display Interface

13.1. Overview

The NGC Real-Time Display application is launched with the following command line:

```
ngcrtd [...options...]
```

This starts the RTD application and the underlying data transfer task, which interfaces to the acquisition process:

-nx <pixels>	Maximum number of pixels in x-direction.
-ny <pixels>	Maximum number of pixels in y-direction.
-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.
-server <name>	Data transfer task server name.
-camera <name>	Unique RTD camera name. A default value for this is given in the ***RTD_CAMERA*** environment variable.
-appname <name>	Unique application name.
-ngcrtdRoot <name>	Database root (default is "*<alias>ngcdcs*").
-ccs	Enforce data transfer task CCS registration.
-noccs	Suppress data transfer task CCS registration.
-verbose <0|1>	Switch verbose mode on/off.

Table 17 NGC RTD Command-Line Options

13.2. How to create a new NGC RTD Data Transfer Task

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

A new data transfer task is created by deriving from the NGC RTD data transfer task base class ***ngcrtdDTT***:

```
class xxrtdDTT: public ngcrtdDTT
{
public:
  // Constructor(s)
  xxrtdDTT(){}

  // Destructor
  virtual ~xxrtdDTT(){}

  // Post-processing call-back
  int PostProcCB(void *, int, rtdIMAGE_INFO *);

protected:

private:
};
```

Then a main process has to be created according to the following template (this is based on the "*ngcrtd/templates/xxrtd.C*" program). The process name can then be passed to the ngcrtd application with the "*-server <process name>*" command line option (see Table 17).

```
/* Post-processing call-back */
int xxrtdDTT::PostProcCB(void *buffer, int frameType, rtdIMAGE_INFO *info)
{
  char tmpBuf[64]; char ermsLoc[256];
  USE(buffer);

  if (ngcppGetFrameNameByType(frameType, tmpBuf, ermsLoc) != ngcppERR_NO_ERR)
    {
    strcpy(tmpBuf, "unknown");
    }
  Verbose("xxrtdDTT: frameType = %d (%s)\n", frameType, tmpBuf);
  return (ngcbSUCCESS);
}

/* Control server process */
static int serverProcess(int argc, char **argv)
{
  xxrtdDTT server;
  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);
      }
    }

  // Intialize
  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. 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. The engineering GUI is launched with the following command line:

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

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

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

In both cases the command line options have the same meanings as the respective ones for the control server (see Table 1). A startup script is also provided (see section 3.2.2).

The tool for editing the controller electronics system configuration is started with the following command line:

```
ngcguiCfg [-cfg <file-name>] [-verbose <level>]
```

15. The NGC General Purpose Control Server

The ***ngcdcs*** base SW module already provides a control server instance ***ngcdcsEvh*** ("*ngcdcs Event Handler*") for general purposes (e.g. for hardware development and tests or to implement command driven sub-systems for controller electronics - see 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). Actually both ***ngcdcsEvh*** and ***ngcircon*** use the same basic server 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 system   (NGC system parameters)
                            |--o exposure (exposure parameters)
                            |--o mode     (read-out mode parameters)
                            |--o guiding  (guiding parameters)
                            |--o seq_<i>  (sequencer parameters)
                            |--o cldc_<i> (CLDC parameters)
                            |--o adc_<i>  (ADC module parameters)
                            |--o acq_<i>  (acquisition module parameters)
```

The database classes for the implemented branches are the same as described in section 6.

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 (see [RDXX]).

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 given by the sequencer program (see section 16.2). The duration of each state (dwell time) in a clock-pattern is defined in the state itself. The value is given in ticks (10 ns). 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***). There are two uploading methods for clock pattern definitions (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.

16.1.1. Binary Clock Pattern Definition

```
# <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
```

16.1.2. 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  "1,2,3,33";   # Mapping list
DET.CLK.MAP2  "37,4";       # Mapping list

# Clock pattern definitions

DET.PAT1.NAME  "Delay";
DET.PAT1.NSTAT 5;
DET.PAT1.CLK1  "00000";
DET.PAT1.CLK2  "00000";
DET.PAT1.CLK3  "00000";
DET.PAT1.CLK4  "00000";     # Convert
DET.PAT1.CLK5  "00000";     # Start pulse
DET.PAT1.CLK6  "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  "0000";
DET.PAT2.CLK3  "0000";
DET.PAT2.CLK4  "0000";      # Convert
DET.PAT2.CLK5  "0110";      # Start pulse
DET.PAT2.CLK6  "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  "0000";
DET.PAT3.CLK3  "0000";
DET.PAT3.CLK4  "0110";      # Convert
DET.PAT3.CLK5  "0000";      # Start pulse
DET.PAT3.CLK6 "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...
```

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
      EXEC <1_second_delay_pattern_1s> $DET.EXPTIME
      JSR <readout>
END
RETUIRN

<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(<readout>)) * 10.0e9

LOOP $N
      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.

The evaluation of arithmetic formulas at run-time is implemented via the TCL scripting language (this is the current VLT software standard and 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. Normally subroutines need not to be declared. The ***SUBRT*** statement is only required in case the execution time of such a subroutine is needed in the evaluation section. The same applies to the parameter list. Parameters which are not used in the script but which are directly referred in the program code also need not to be declared.

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. It has to be considered that all (local and global) floating-point parameters which are used in the **program-code** are rounded towards the nearest integer value.

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 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|$paramater_name> <repetition_factor|$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
```

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. Voltage Setup

The voltage setup is done through a configuration file in short FITS format. 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 a setup command using the same keywords (***DET.CLDCi.CLKHIi***, ***DET.CLDCi.CLKLOi***, ***DET.CLDCi.DCi*** for setting the clock- and the bias-voltages, ***DET.CLDCi.CLKOFF***, ***DET.CLDCi.DCOFF*** for setting the offset voltages). The current voltage setup can be saved again to a file in this format (***CLDC –save <file>*** command, see section 4).

```
##############################################################################
# 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)
#
##############################################################################

# Offsets:
DET.CLDC.CLKOFF 10.0;   # Global clock voltage offset
DET.CLDC.DCOFF  10.0;   # Global DC voltage offset

# 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 16 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 20 DC-voltages like this ...
```

16.4. 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 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***).

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.NAME	->	*DET.READi.NAME*
DET.IRACE.RMi.ACQi	->	*DET.READi.ACQi*
DET.IRACE.RMi.SEQ	->	*DET.READi.SEQi*
DET.IRACE.RMi.DSUP	->	*DET.READi.DSUP*

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/
```

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*>). An editing tool for the system configuration (***ngcguiCfg***) is provided by the ***ngcgui*** module (see [AD9] and section 14).

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

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 in the ***DET.SEQi.DEVIDX***, ***DET.CLDCi.DEVIDX***, ***DET.ADCi.DEVIDX*** keywords and a linking route *through* the interface *to* the target module. 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).

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

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

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

The system configuration file finally holds additional keywords for specifying a default detector configuration file, for specifying default values for the file formats and the naming scheme and for some general system behavior like starting the sequencer(s) automatically when going to ***ONLINE***-state (***DET.AUTOSTRT*** = "***T***"). By default the voltages on the CLDC modules are not enabled when going online and also the acquisition is not started. This behavior can be changed by setting ***DET.CLDCi.AUTOENA*** to "***T***" or "***F***".

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.

Example of controller electronics system configuration 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

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

# 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)

# 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      "$HOST";         # host name for acq.-process
DET.ACQ1.CMDPORT   0;               # acq.-process command port (optional)
DET.ACQ1.DATAPORT  0;               # acq.-process data port (optional)
DET.ACQ1.NCLIENT   2;               # max. number of data server clients
DET.ACQ1.SEQIDX    1;               # associated sequencer instance

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

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*>). If 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 this chip to a certain acquisition module (***DET.CHIPi.ACQIDX***). This is required to pass the right frame dimensions and window parameters to the associated acquisition process. A zero index means that the chip definition applies to all acquisition modules.

The detector configuration file defines the clock pattern configuration files, which have to be loaded into the sequencer module(s) for this detector/mosaic (***DET.SEQi.CLKFILE***), and also the default values for the global state 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. will react on the external run-signal). The detector voltage configuration files (***DET.CLDCi.FILE***) have to be specified for all CLDC modules which are used in this detector configuration. Not used CLDC modules can simply be skipped and their outputs will not be automatically enabled together with the other ones.

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

The detector configuration also declares the read-out modes to be used with the given chip(s) (see section 16.4).

Example of Detector Configuration File:

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

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

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

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

# ADC module setup
DET.ADC1.DELAY      0;            # conversion strobe delay (ticks)
DET.ADC1.OFFSET     2.0;          # ADC offset (Volt)
DET.ADC1.ENABLE     4;            # number of enabled ADC units
DET.ADC1.OPMODE     0;            # ADC operation mode
DET.ADC1.SIMMODE    0;            # ADC simulation level
DET.ADC1.PKTSIZE    0;            # packet size (0 = let system choose)
DET.ADC1.CONVERT1   "T";          # convert on strobe 1
DET.ADC1.CONVERT2   "F";          # convert on strobe 2

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

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

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

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. Configuration for HW-Control

In this case the controller electronics system configuration does not define any acquisition module (i.e. no ***DET.ACQi*** category keywords, the ***DET.FRAM*** category keywords are ignored and can 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 sections 6 and 15) exist, but are not in use when the system is configured for HW-control only.

17.3.2. Configuration for Data Acquisition

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. 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"
#undef ngcirconROOT
#undef ngcirconINSTANCE

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

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 I^{2}C Bus Interface

In order to communicate with other sub-devices (e.g. the pre-amplifier board) the NGC DFE implements a two wire I^{2}C 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 I^{2}C Interface Class

The bus protocol timing is implemented via a general I^{2}C 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 I^{2}C 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 I^{2}C standard. It is also possible to ignore the acknowledge signal at all.

The I^{2}C 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. For trivial communication the acknowledge bit may be ignored.

18.2. Engineering Interface

The driver interface process takes care of the I^{2}C bus protocol timing whenever the I^{2}C 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 placed on bit 2. The clock-stretching capabilities are disabled 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 I^{2}C 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 18) and the I^{2}C 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.

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 18 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 19 I^{2}C Protocol Tuning

<u>Examples</u>:

Write one 16-bit value to the I^{2}C 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^{2}C bus expander):

```
"wraddr 0x2 0x31C021 0xABCD"
```

Write three 8-bit values to I^{2}C 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^{2}C 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^{2}C 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^{2}C bus expander):

```
"wraddr 0x5 0x5 0x2 0x2C020 0xABCD"
```

19. 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.

19.1. How to create a new Server

A new 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/
```

19.2. 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.

19.3. 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.

19.4. 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
int scaleFactor;    - Scaling factor to be applied to normalize
int bitPix;         - Bits per pixel as defined in the FITS-standard
int sx;             - Lower left corner (x-direction)
int sy;             - Lower left corner (y-direction)
int nx;             - Dimension in x-direction
int ny;             - Dimension in y-direction
double crpix1;      - Reference pixel in x-direction
double crpix2;      - Reference pixel in y-direction
int detIdx;         - Detector index (for mosaics)
int expCnt;         - Exposure counter for this type
char utc[64];       - Time when frame was ready in the pre-processor
ngcdcsCUBE *cube; - Data cube object to be used for storing to a cube
```

The ***ngcdcsCUBE*** class contains the following members:

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

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

<u>CAUTION:</u>

Because of its concurrency with the event handler the post-processing call-back must not send or receive CCS-messages. The online database and the most other VLTSW modules (like slx, oslx) are not affected by this restriction. All CCS-compliant functions executed inside the call-back must use the *error* stack, which is passed as parameter to the callback.

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.
DET.SHUTi.REGSTAT   - Shutter status register. The STATUS command returns
                      a hexadecimal value (0x…).
DET.SHUTi.EXPTIME   - Exposure time register (decimal value).
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 [AD8] for a detailed description of the register contents and functionalities.

21. Chopping Mode

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

The computation of the result image (*object* - *sky*) is application specific. Usually the parameters ***DET.CHOP.NCYCLES*** and ***DET.CHOP.CYCSKIP*** define the number of chopping cycles and the number of cycles to skip after start, and ***DET.IR.NDIT*** + ***DET.IR.NDITSKIP*** integrations will be done on each chopping half cycle. The ***DET.IR.NDITSKIP*** integrations are skipped at the beginning of each half cycle. Optionally the data for the chopping half cycles can also be computed (controlled via the ***FRAME***-command - see section 9.5). However - these parameters are application specific and may not be valid in all cases. If applicable they will be available either via the ***STATUS***-command or via the parameter table in the database attribute ‘***<alias>ngcircon:system.param***’.

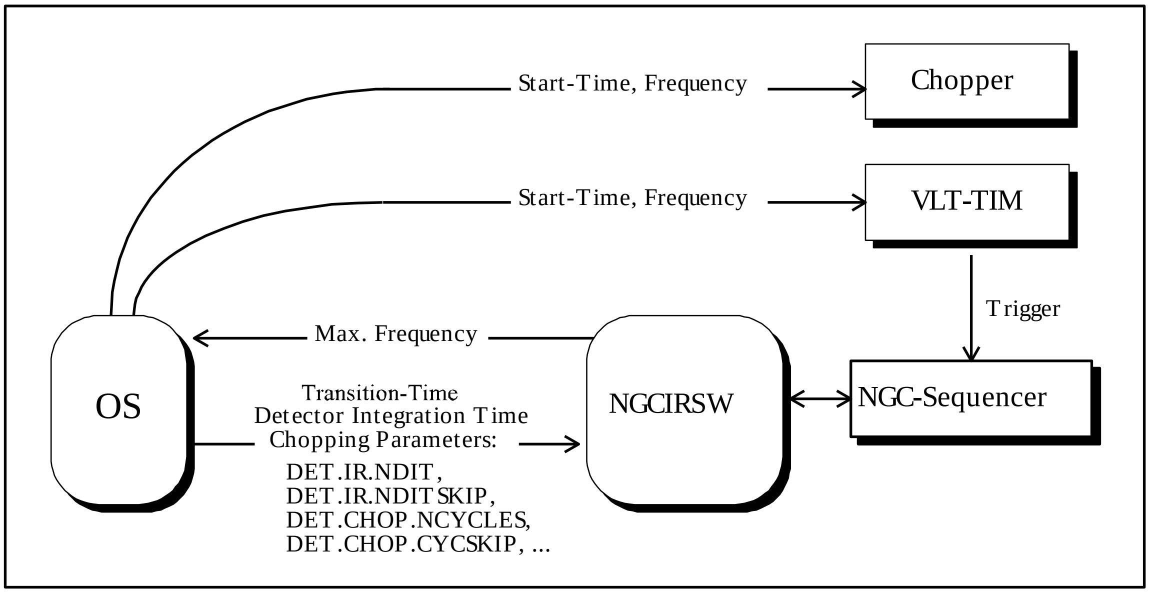

Figure 4 Chopping Mode

Changes with respect to IRACE:

The handling of the chopping mode is backwards compatible with IRACE. The database point and attributes have changed.