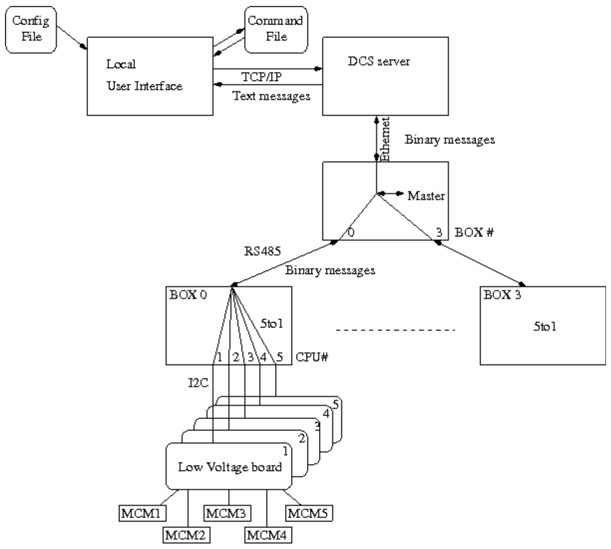
DCS text message format

version 20140522 (updated 20140605)

This document describes the text messages sent between a user interface and the DCS server. The DCS server handles the communication with the master of the DCS. The layout of the system is shown in the figure below.



Header

Each message must have a header containing destination and instruction. The parameters must follow after the header.

SOURCE.TYPE	1

The type of source to send any reply to. Currently must the argument be set to 1. This will be used to distinguish between different clients connecting to the DCS server.

DESTINATION.BOX	<BOX number>
DESTINATION.BOX.CPU	<CPU number>

The destination is given by the BOX number (0-3) and the CPU number 1-5, 0 = all) within
a box. These lines must be first in the message. A CPU number = 0 is only allowed when no data is expected to be sent back.

INSTRUCTION.<type>	<0-disable/1-enable>

There are different types of instructions. Only one instruction in a message may be enabled. Not all can be used in a text message.
<type> can be:

ACK acknowledge a received message

This is sent automatically by the DCS server to the master after having received a SCANEND message. This will be forwarded by the master to the 5to1 CPU in the destination given in the header. The cpu will when receiving the ACK update what was last sent to the master. *This can not be used in a text message.*

ALARM get status of limit counters that may cause an automatic power off

The 5to1 CPU is scanning, with some interval (here called cycles), the temperatures and ADCs that monitors the voltages. If the temperature or currents exceed a limit will two counters be incremented. One counter is incremented each time when the limit is exceeded. The second is incremented (can not go above a maximum number of cycles) if the limit is exceeded and decremented (can not go below 0) if it falls below the limit again. If this counter reaches a maximum number of cycles then the hardware is powered off. With this instruction are these counters read.

CHANGES changes since last values sent

This is automatically sent by the master when scanning BOX/CPU for ADC values. The CPU will only send read ADC values that differ with a given amount from what was last acknowledged with an ACK. *This can not be used in a text message.*

LOAD load new settings

change the settings as given in the parameters in the text message body

READ read settings or values from hardware

read settings in the master or cpu, or read values from hardware. What to read is given in the text message body.

SCANEND last box/cpu has been scanned

This is sent from the master to the DCS server when it has finished a scan of all CPUs. The master scan with an interval (cycles) the CPUs for temperatures and voltages as read from sensors and ADCs. *This can not be used in a text message.*

STATUS return status of items

Get status of variable parameters. The parameters are given in the text message.

Instructions that can be used in a text message header are:
ALARM, LOAD, READ, STATUS

Instructions that can NOT be used in a text message header are:
ACK, CHANGES, SCANEND

Text message body

Each line is formatted as:
<part0>.<part1>.<part2>		<value>

<part2> is not always required.

Power handling

Voltages are switched on/off through IO registers, one register for each MCM.

<u>*Switch on/off regulators*</u>

IO.MCMn.VOLTAGEm		*b*
Instruction = LOAD
n = MCM number 1-5
m = Voltage regulator 0-7
b = 0: switch off, 1: switch on

reply:
	the same is returned

<u>*Read current setting in register*</u>

IO.MCMn		*b*
Instruction = READ
n = MCM number 1-5
b = 0: do not read, 1: read current value of IO register for this MCM

reply from CPU:
	IO.MCMn.VOLTAGEm		*b*
	Instruction = READ
	n = MCM number 1-5
	m = Voltage regulator 0-7
	b = 0: switched off, 1: switched on

Voltage/Current monitoring

The voltages to the regulators and from the regulators are monitored with ADCs.
There is a resistor before each regulator, by measuring the voltage drop across the resistor can the current be calculated.

<u>*Read ADC values for voltages IN to all regulators*</u>

ADCIN.ALL.POWERm		*b*

Instruction = READ
ALL = the voltages supplied to the regulators
m = 1-8: the channel in the corresponding ADC
b = 0: do not read this channel, b = 1: do read this channel

reply:

> *ADCIN.ALL.POWERm* *v*
> Instruction = READ
> ALL = the voltages supplied to the regulators
> m = 1-8: the channel in the corresponding ADC
> v = value in ADC counts (=mV)

Read ADC values for voltages IN to individual regulators

ADCIN.MCMn.POWERm *b*
Instruction = READ
n = The voltage after the resistor to MCM 1-5
m = 1-8: the channel in the corresponding ADC
b = 0: do not read this channel, b = 1: do read this channel

reply:

> *ADCIN.MCMn.POWERm* *v*
> Instruction = READ
> n = The voltage after the resistor to MCM 1-5
> m = 1-8: the channel in the corresponding ADC
> v = value in ADC counts (=mV)

Read ADC values for voltages OUT from individual regulators

ADCOUT.MCMn.POWERm b
Instruction = READ
n = The voltage after the regulatos to MCM 1-5
m = 1-8: the channel in the corresponding ADC
b = 0: do not read this channel, b = 1: do read this channel

reply:

> *ADCOUT.MCMn.POWERm* *v*
> Instruction = READ
> n = The voltage after the resistor to MCM 1-5
> m = 1-8: the channel in the corresponding ADC
> v = value in ADC counts (=mV)

Temperature monitoring

On each MCM and on the Low Voltage boards are temperature sensors.

Read temperature

TEMP.MCMn.VALUE *b*
Instruction = READ
n = The sensor on MCM 1-5
b = 0: do not read this sensor, b = 1: read this sensor

reply:

> *TEMP.MCMn.VALUE* *v*
> Instruction = READ
> n = The sensor on MCM 1-5
> v = value read in raw data format

TEMP.LV.VALUE *b*
Instruction = READ
LV = Sensor on Low Voltage board
b = 0: do not read this sensor, b = 1: read this sensor

reply:

> *TEMP.LV.VALUE* *v*
> Instruction = READ
> LV = Sensor on Low Voltage board
> v = value read in raw data format

<u>Read and write LOW and HIGH registers in sensors</u>

In the sensor are register for LOW and HIGH values of temperature that can be used to set a hardware signal from the sensor. *This is not used but the register can be written and read.*

TEMP.xx.LOW b
Instruction = READ
xx = sensor as above
b = 0: do not read this sensor, b = 1: read this sensor

reply:

> *TEMP.xx.LOW* *v*
> Instruction = READ
> xx = sensor as above
> v = value read in raw data format

TEMP.xx.LOW *v*
Instruction = LOAD
xx = sensor as above
v = value to write in raw data format

reply:

> same as written

TEMP.xx.HIGH *b*
Instruction = READ
xx = sensor as above
b = 0: do not read this sensor, b = 1: read this sensor

reply:

> *TEMP.xx.HIGH* *v*
> Instruction = READ
> xx = sensor as above
> v = value read in raw data format

TEMP.xx.HIGH *v*
Instruction = LOAD
xx = sensor as above
v = value to write in raw data format

reply:

> same as written

DAC settings

On each MCM is a DAC, which is used for various reference voltages for the SALTRO.

Read current values in DAC

DAC.MCMn.CHANNELm *b*
Instruction = READ
n = MCM 1-5
m = 1-8: the channel in the corresponding DAC
b = 0: do not read this channel, b = 1: do read this channel

reply:

> *DAC.MCMn.CHANNELm* *v*
> Instruction = READ
> n = MCM 1-5
> m = 1-8: the channel in the corresponding DAC
> v = value read from this channel

Write values to DAC

DAC.MCMn.CHANNELm *v*
Instruction = LOAD
n = MCM 1-5
m = 1-8: the channel in the corresponding DAC
v = value to write to this channel

reply:

same as written

Scanning

The CPUs can be configured to scan temperature sensors and ADCs. The scan intervals can be set separately for temperature sensors and ADCs. If the temperature or current exceeds configurable limits is the power automatically switched off. The current limit is set for each regulator, while the limit is the same for all temperature sensors.

Since the CPUs only send information on request must the master be configured to scan CPUs for last read temperature values and changes in ADC readings. The intervals can be set separately for sensors and ADCs.

<u>*Set scan time periods*</u>

SCAN.TEMP.PERIOD	*v*
SCAN.ADC.PERIOD	*v*

Instruction = LOAD
v = scan temperature and ADC with the period in seconds

reply:

same as written

<u>*Read scan time periods*</u>

SCAN.TEMP.PERIOD	*b*
SCAN.ADC.PERIOD	*b*

Instruction = READ
b = 0: do not read, 1: read the scan period of temperature and ADC

reply:

SCAN.TEMP.PERIOD	*v*
SCAN.ADC.PERIOD	*v*

Instruction = READ
v = scan temperature and ADC period in seconds

Shutdown limits

Upper limits on currents and temperature are set by the LIMIT message. If the limit is exceeded for CYCLES number of scan intervals is the power switched off. The algorithm to switch off is that if the limit is exceeded is a counter incremented (up to the CYCLES value), and if less than the limit it is decremented (if the counter is greater than 0). When the counter reaches CYCLES is the power switched off. The MCMs are handled individually. If the LV temperature gets too high are all MCMs switched off.

<u>*Set and read shutdown cycle values*</u>

LIMIT.CURRENT.CYCLES	*v*
LIMIT.TEMP.CYCLES	*v*

Instruction = LOAD
v = number of periods above shutdown limit before shutdown is done

reply:
same as written

LIMIT.CURRENT.CYCLES	*b*
LIMIT.TEMP.CYCLES	*b*

Instruction = READ
b = 0: do not read, 1: read number of periods above shutdown limit before shutdown is done

reply:

LIMIT.CURRENT.CYCLES	*v*
LIMIT.TEMP.CYCLES	*v*

Instruction = READ
v = number of periods above shutdown limit before shutdown is done

<u>Set and read shutdown limits</u>

LIMIT.CURRENT.POWERn	*v1*
LIMIT.TEMP.SHUTDOWN	*v2*

Instruction = LOAD
n = power 1-8
v1 = shutdown limit in milliAmpere
v2 = shutdown limit in degrees

reply:
same as written

LIMIT.CURRENT.POWERn	*b*
LIMIT.TEMP.SHUTDOWN	*b*

Instruction = READ
n = power 1-8
b = 0: do not read, 1: read shutdown limits

reply:

LIMIT.CURRENT.POWERn	*v1r*
LIMIT.TEMP.SHUTDOWN	*v2*

Instruction = READ
n = power 1-8
v1r = shutdown limit in milliAmpere
v2 = shutdown limit in degrees

Note that v1r may differ from v1 written since it is converted to a voltage difference, using the resistor value over which the voltages are measured, by the server before sent to the hardware. When read back the voltage difference is converted back to a current by the server.

SHUTDOWN handling

Status of automatic shutdown

FAIL.MCMn.TEMP	*b*
FAIL.LV.TEMP	*b*
FAIL.MCMn.ADC	*b*

instruction = STATUS
n = MCM 1 to 5
b = 0: do not read, 1: read shutdown status due to temp (TEMP) or current (ADC)

reply:

FAIL.MCMn.TEMP	*b*
FAIL.LV.TEMP	*b*
FAIL.MCMn.ADC	*b*

instruction = STATUS
n = MCM 1 to 5
b = 0: not shutdown, 1: shutdown done

Note that this information is always sent by master when scanning, together with values scanned.

Clear shutdown status

FAIL.MCMn.TEMP	*b*
FAIL.LV.TEMP	*b*
FAIL.MCMn.ADC	*b*

Instruction = LOAD
n = MCM 1-5
b = 0: do not clear, 1: clear shutdown status

reply:

FAIL.MCMn.TEMP	*b*
FAIL.LV.TEMP	*b*
FAIL.MCMn.ADC	*b*

Instruction = LOAD
n = MCM 1-5
b = 0: not shutdown, 1: shutdown done

Note that this will also clear the shutdown counters.

Current values of shutdown counters

LIMIT.MCMn.TEMP	*b*
LIMIT.LV.TEMP	*b*
LIMIT.MCMn.POWERm	*b*

instruction = ALARM
n = MCM 1-5
m = POWER 1-8
b = 0: do not read, 1: do read the current values of shutdown counters

reply:

LIMIT.MCMn.TEMP	*v*
LIMIT.LV.TEMP	*v*
LIMIT.MCMn.POWERm	*v*

instruction = ALARM
n = MCM 1-5
m = POWER 1-8
v upper byte = total number of cycles above limit
v lower byte = current number of consecutive cycles above limit