Zdanetix

The versatile Home Automation system

Reference Manual

Helmut Müller

Version x2
Second Draft, December 2015

http://www.achemtronic.eu

CONTENTS

The **Zdanetix** project (Russian **здание**, *building*) is an effort to develop a simple, multifunctional Harware/Software home automation application, that can be used for small business premices or private houses.

Its main objectives are as following :

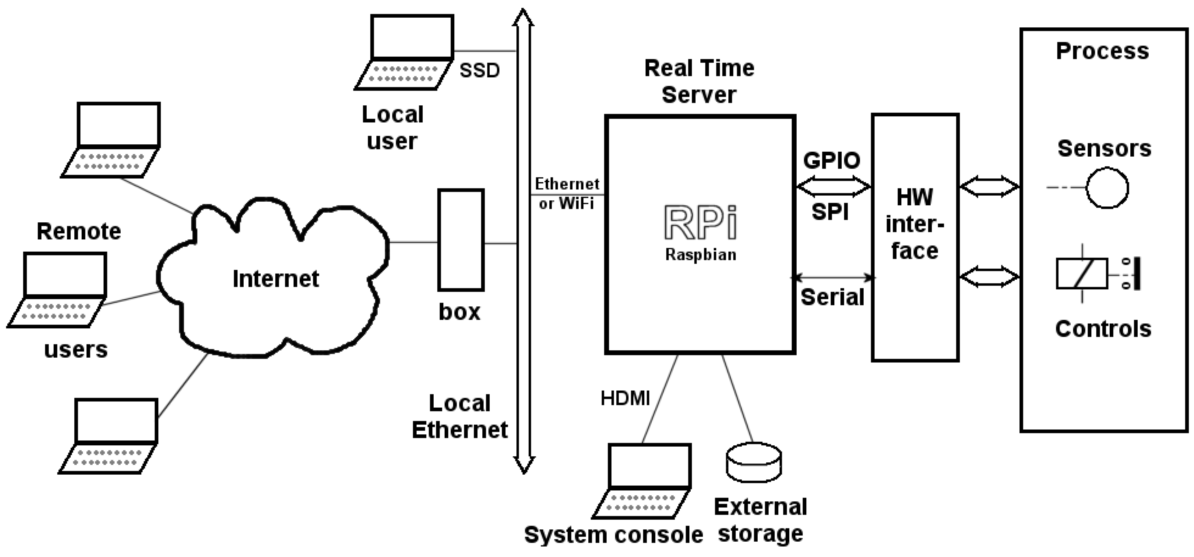
- acquire, record and process a limited number of numeric and analogic data (a few dozens)
- elaborate numeric and analogic controls while combining these data with commands and setpoints entered by the user
- heating, lighting, access, etc. control and/or monitoring
- data logging and journalling
- provide a friendly user interface accessible through the Internet, using a web browser, without the need to install additional software on the client side
- be simple and open enough to be easily portable to small low-end configurations.
- be localization friendly, at least for the languages using an 8-bit latin alphabet

The web site http://www.achemtronic.eu is largely dedicated to this software and presents a demonstration of an emulated subset of the user interface.

1. Presentation

1.1 General configuration

For performance and portability reasons, the software is written in C. It can be ported to any system configuration with adequate computing resources (CPU, memory, mass storage, system software) and a decent C compiler and library. It runs well on a Windows system, which is its current development environment (the free version of Visual Studio), but, because of a combination of features usually not available on most other systems, the preferred deployment platform is the **Raspberry Pi**, model B or B+ (abbreviated as RPi).
Therefore, unless stated otherwise, any further reference to non-standard hardware features (GPIO, etc.) will refer to this platform.



The overall configuration implements a real-time process control system featuring :

- on one side : multiple connections to sensors and controls of a specific process, which are managed by a dedicated *hardware interface* that connects to one or several of the computer's I/O ports (USB, COM, SPI, GPIO, etc.)
- on the other side : a web server connected to a local Ethernet and/or the Internet, providing the navigator based user interface
- in the middle : the application software.

A local system console, which may be useful for privileged maintenance transactions, is optional in normal operation and can be emulated on a PC through the network.
Video and sound functionalities are not currently supported by the software.

1.2 The Raspberry Pi in short

The Raspberry Pi (while keeping an eye on some of its recent competing clones) presents a unique combination of features that makes it a platform of choice for home control applications :

- its low cost, which encourages to deploy several of them, each dedicated to a specific small application or an independent subset of the whole process, avoiding the necessity to overload one device with too much functionality or activity
- its adequate performance for relatively slow and low-end process control applications
- its small footprint (physical dimensions and power requirements)
- its embeddability
- its large and evergrowing user community

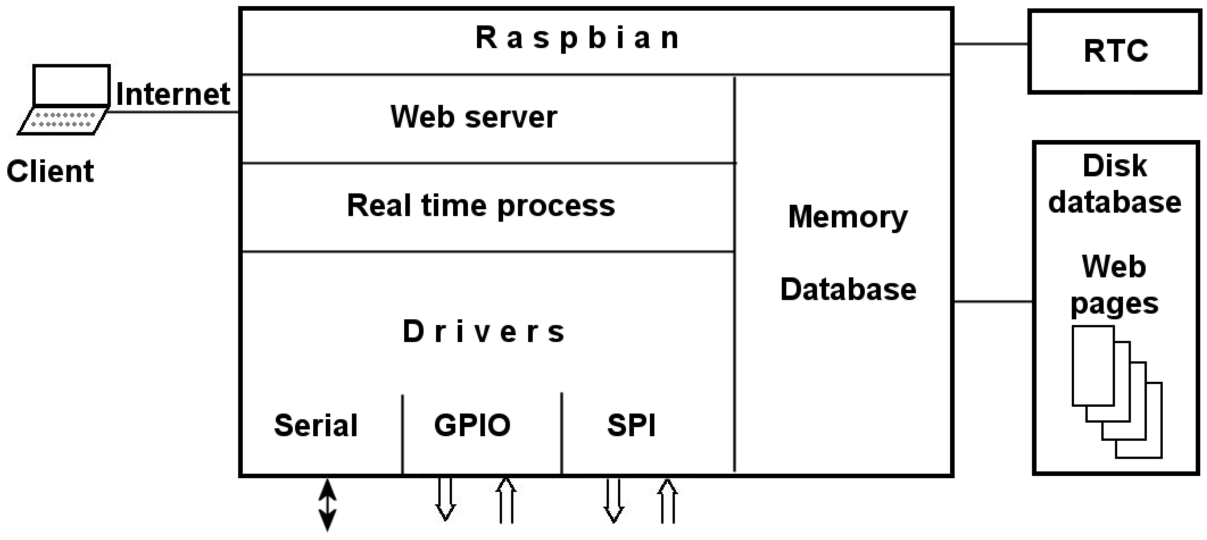
Besides its Ethernet port, the feature of interest is its GPIO port, with its various subfunctions :
- a serial port, connectable to a microcontroller used as process control connections expander
- an I2C port, interfacing a real-time clock module
- a SPI port, connected to I/O expanders and ADC inputs
- individual GPIO pins, used for specific functions like providing addresses to multiplexors

However, it lacks some important features which must be provided by the interfacing hardware :
- analogic I/O
- a real time clock, if time must be preserved across reboots and losses of Internet connection

There is already a wealth of literature and Internet information available on the Raspberry Pi and Raspbian, therefore the present document will only talk about whatever is necessary to work with the application.

Note : When referring to GPIO pins in the following, except otherwise stated, the BCM2835 pin designations are used, not the Raspberry Pi pin header layout.

1.3 Overall structure of the software



The application software is built around a database stored on external mass storage (disk or SD card for the RPi) with a subset in memory, for the information that must be immediately available. This subset is built up from scratch during the startup phase.
Except for the operating system (Linux Raspbian for the RPi), in this version, the application software is bundled in a single executable with the following functions :

- a web server executing several identical threads
- an interface between the web server and the data base, similar to a CGI or PHP procedure in standard web server configurations
- a process management loop executing as a unique background thread, which does all the data processing and calls the drivers.
- drivers that support the various I/O protocols

1.4 Terminology

This is a list of terms used in this document, along with their accepted meanings. Some of them may have more than one meaning, depending on the context. If a term refers to any of those, the referred meaning is specified in parentheses.

Application	combination of the *system* and the *process(1)*
Bank	A group of 8 *numeric channels* mapped to one byte of memory. These channels are numbered *8n* to *8n+7,* with *n=0..7*
Channel	unit of data interchange with the *process(1)* *Numeric or digital channel* 1-bit value, 0 or 1 *Analogic channel* multi-bit value (usually expanded to an integer number of bytes) *Input or output channel* defines the direction of the data interchange. Set up on a per channel or per *bank* basis, depending on the hardware and the needs of the *application* *Virtual channel* invisible to the hardware, used by software as both input and output *Hidden channel* only visible to the hardware for hardware ancillary functions *Channel number* a number uniquely identifying the channel, like an address *Channel name* a text string associated with a channel, usually telling its usage
Computer	the part of the *system* that runs the software, manages the data and supports the *user interface*
Current value	a value supplied by the *process(1)* at a given time *(now)*. May be *raw* or *physical* (*French:* mesure, *German:* Istwert). Frequently compared to a *setpoint.*
Data base	collection of data used by the *system*, both on *disk* and memory
Disk	catch-all name for whatever external mass storage is available on the *system* : flash memory, USB key, SD card, hard disk, etc.
Drivers	part of the *software* that drives the *hardware interface.*
Hardware	loose term for the *Hardware interface*, the *Drivers* or both, depending on context
Hardware interface	the hardware subsystem that interconnects the *system* and the *process(1)*
I/O matrix	Structure in main memory containing the *raw values* of all *channels*
Node	1. record on *disk* containing the values of *system parameters* 2. structure in memory initialized wih those values
Physical value	a *raw value* converted to a value in some physical unit. For *numeric values,* this may be the *raw value* or its complement For *analogic values,* this may imply a more or less complicated transformation of the *raw value* (currently only linear)
Presentation value	a *physical value* after a *rescaling* designed to display or print it in a meaningful unit, format and range
Process	1. the environment controlled by the *system*; 2. part of the application software that manages the *process(1)*, not including the *drivers* 3. a unit of activity in the computer (may be used for a thread)
Program	1. a subset of the *software(1)* 2. a collection of values, usually organized on a time-of-day basis, which are used as timed *setpoints* 3. a read-only list of statements executed by a *sequence*
Quarter hour	abbreviation for quarter of an hour, a 15 minutes time interval. Within a day, they are numbered from 0 to 95. This is the shortest time period for daily *programs(2)*
Raw or **hardware value**	any value exchanged between the *computer* and the *hardware interface*
Rescaling	the operation of multiplying or dividing a value by a scaling factor and possibly applying an offset. This may be needed for storage or display requirements and is done identically for all values of a given type.

Sequence	a unit of execution of a *program(3),* allowing to combine logically the values of input and output *channels* for executing timed and sequential actions on the *process(1)*
Setpoint	a value set by the *user* or computed by the *system* and used as reference (*French:* consigne, *German:* Sollwert). Frequently compared to a *current value.*
Software	1. the collection of application programs, including the *process(2)* 2. the data management and the *user interface*
State word	one of the entries in a system wide *state words table* contained in a Javascript module included by HTML pages using this feature, and used as a channel state indicator and in messages telling the occurrence of an event on a channel. Most used state words : *on, off, open, closed, low, high* ...
Stored value	the *physical value* as stored on *disk*. Possibly subject to *rescaling* to fit the size of a storage cell
System	the hardware/software combination that controls a *process(1)* (includes the *computer* and the *hardware interface*)
System *or* **configuration parameter**	system wide value. Usually initialized at startup from a value on disk (see *node*), but may be hard coded or modifiable on-line (exceptionally, for debugging / testing).
User	1. person that interacts with the *application*; 2. user agent (browser)
User interface	part of the *software* supporting the *user(1)* interaction with the *application.* Mostly implemented by a collection of dynamic web pages.

A *channel* can be *numeric* (term used interchangeably with *digital*) or *analogic* (sometimes abbreviated to *analog*). The primary identification of a channel is its *channel number*. The software supports a *channel space* of 128 channels, where channels 1 to 63 are numeric, and channels 64 to 126 are analogic. 0 and 127, although implemented in the data structures, are reserved values and cannot be used as actual channel numbers (0 usually means 'no such channel' or 'take a default').

The input or output direction of a numeric channel is a priori undefined, until it is configured by the user or dictated by the layout of the hardware interface (in which case the user configuration must obviously follow).

The data flow direction of analogic channels is set by the hardware which usually incorporates ADC or/and DAC circuits connected via an I2C or SPI bus. The current version of the sofware provides support for the MCP3xxx ADC series on the Raspberry Pi. Extending the support to other devices would require the addition of dedicated drivers.

A channel may be either *physical*, that is associated with an actual hardware I/O channel, or *virtual*, dedicated to some special internal use by the software, in which case it may act as both input and output, as its value is fully controlled by the software.

A *physical channel* (usually numeric) may be *hidden*, that is invisible to the *process(2)*, and reserved to some hardware management task like multiplexor switching.

1.5 The database

1.5.1 Overview

The system's **data base** has one part in active memory (RAM) and another part on external mass storage (disk or SD card).

The part on external mass storage is divided into independent areas called **domains**, each containing a specific type of data : *programs(2),* channel tables, channel names, message texts, journals etc.
Every domain is identified by a ***domain letter*** (A-Z), some domains (those that are user editable) by a name. Domains are spread across a number of physical files and are random access, divided into constant length records, subdivided into variable length fields (all records in a domain have the same structure). Two domains may map the same storage space, allowing different views of the data.
The standard record size is currently 128 bytes (the same for all by convenience, not by necessity).
The contents is basically 8-bit text, the contents of a byte being occasionally interpreted as a binary value.

The part in active memory includes the input/output matrix (**I/O matrix**) and the **memory channel tables**.
These structures are set up dynamically at startup time.

The I/O matrix contains an image of the raw values of all channels and is used for data interchange with the process. It supplies the current values for input channels processing and holds the setpoints for output channels.
The memory channel tables contain binary copies of a subset of the channel tables on disk and dynamic values read from the process (mainly the raw and/or the physical value) or resulting from processing. The tables are organized as a linked list, ordered by priority. In order to optimize processing speed and memory usage, only the channels specifically configured by the user have a table in memory (the others have no specific processing or are unused).

1.5.2 The domains

The domains are designed independent in size and structure. Each domain is associated with a data structure that specifies its area on disk, its size in records and the subdivision of its records into fields. Short access time being a significant requirement, random access was preferred and a same constant record size was set to 128 bytes. As the number of records in a domain is frequently related to the number of channels and some waste of storage space is no longer a problem with current disks or SD cards, all domains (except system and journal domains) have a size of 256 records.
When the software requires that the same data area be viewed as two different structures (one for editing and one for processing), this area is mapped by two different domains.

List of domains

Letter	Editable	Contents
@	yes	Privileged system access *(overmaps the whole database)*
A	**yes**	Numeric programs full *(overmaps B)*
B	-	Numeric programs 1-byte
C	**yes**	Analogic programs full *(overmaps D)*
D	-	Analogic programs 1-byte
E	**yes**	Channel tables full *(overmaps F)*
F	-	Channel tables by fields
G	yes	User profiles
H	yes	Constants and parameters
I	**yes**	Values for simulation full *(overmaps J)*
J	-	Values for simulation 1-byte
K	yes	Macros
L	-	Sequence Control Block info full *(overmaps M)*
M	**yes**	Sequence Control Block info by fields
N	yes	Sequential programs
O	yes	General names dictionary
P	yes	General messages dictionary
Q	-	Journal 0
R	-	Journal 1
S	-	Journal 2
T	-	Journal 3
U	yes	Numeric and analogic *program(2)* names dictionary
V-Z	-	Reserved

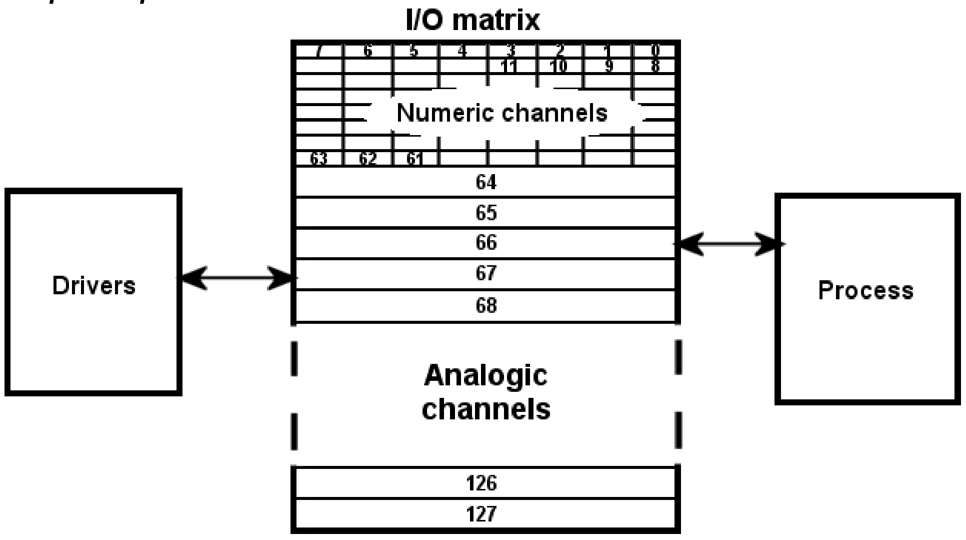
In this list, ‘yes’ in normal face means that editing is performed by the standard ***catch-all editor*** and ‘**yes’** in bold face that it is done by the dedicated ***program editor*** or ***channel table editor.***

1.5.3 The channel tables

Each possible channel is associated with a collection of static and permanent parameters describing its usage, whose values are stored in an individual channel table in the dedicated E *domain* on *disk*. The channel number is used as record number for direct access to this table.
A parameter in this table specifies if the channel is idle or involved in some kind of more or less sophisticated processing, in which case a table, containig a subset of the table on disk and a few additional dynamic variables, is ‘loaded’, that is allocated and initialized in memory, at startup.

A memory channel table cannot be loaded or unloaded at run time : a restart of the application is required.

1.5.4 The input/output matrix



The **I/O matrix** is a permanent central software interface between the *drivers* and the *process(2),* fully disconnecting their respective activities. It is a table holding the *current raw values* of all channels. On one side, the drivers periodically refresh the values of input channels and copy the values of output channels to the *hardware interface*, on the other side, these values are read or written by the process. This allows the process to run even if the hardware is disabled or missing, which is required for development, debugging and the maintenance of the *database*. In addition, it makes the *raw values* easily accessible for display or modification and facilitates the implementation of test or simulation functions and the addition of drivers supporting new interfacing hardware.

2 Configuration

2.1 Configuration parameters

Configuration parameters on disk are scanned and copied to memory at application startup. If they are later modified, a restart is required for the modification to take effect.

There are two categories of configuration parameters:

- The primary options, which must be available at the very start of the application and modifiable without the application running. They are stored in the *configuration file*, a standard text file editable with a text editor.
- the secondary parameters, stored in domain H : « Constants and Parameters ». They require the application to run to be user accessible.

2.1.1 The configuration file

This file, with the file extension `.conf` for Linux and `.confw` for Windows, must be located (or referenced) within the default folder of the application (usually the folder containing the application executable file) as it is the first file accessed by the application. It contains all the information that must be known at application startup. This is a listing of a typical configuration file :

```
# zdanetix.conf - Application configuration file - Linux version
#   ©2015 H. Müller
#
# Edits:
#    1  10-may-15  creation
#    2  07-jun-15  revised for daemon operation
#    3  11-jun-15  PIDFILE
#    4  19-aug-15  OPTMASK
#    5  04-oct-15  'w' log file
#    6  05-oct-15  OPTMASK : bits 0 and 31
#    7  22-oct-15  LOCPATH
#    8  30-oct-15  no trailing slash in option values
#
# Caveat: the Linux version of the config file must be in Linux format (new line=LF),
#         It would fail if in DOS format (new line=CR/LF)
#
# This file declares options that must be available early at startup and are normally
# not modifiable during operation.
# It must be tailored for the system the application is running on (OPTMASK bit 31 is
# provided for detecting the use of a wrong file).
#
# On Linux, its standard location is /etc/zdanetix/zdanetix.conf
# (See also /etc/init.d/zdanetix)
#
# Option declaration syntax:
#
#    option_name option_value
#
# - a single space between name and value
# - all process options are mandatory, web server options have defaults
# - one leading space for process options; no leading space for web server options
# - no space allowed in the option name
# - comment and blank lines are ignored
# - declaration lines cannot have trailing spaces or comment
# - the software expects process options to be declared first and in a specific order:
#     0 PROCROOT application data base path
#     1 LOGFILE  log file name. If first char is :
#                 '_' -> on Windows, closes the launch window (this char is skipped)
#                 '*' -> log on stderr, no forking
#                 '@' -> no log, default on log file open failure
#                 '!' -> append to previous log file (this char is skipped)
#     2 PIDFILE  PID filespec (linux)
#     3 OPTMASK  options enable/disable mask: 0=disabled, 1=enabled
#                bit 0 : privileges granted without login (for testing or secure environments)
#                bit 1 : SPI
#                bit 2 : GPIO
#                bit 3 : ADC
#                bit 4 : SERIAL
#                bit 31: file for Windows
#     4 LOCPATH  path to locale directories
#
# Process options
# --------------------------------
 PROCROOT /var/www/db
 LOGFILE /var/www/web/zlog.htm
 PIDFILE /var/run/zdanetix/zdanetix.pid
 OPTMASK 0x0f
 LOCPATH /var/www/web

# Web server options
# --------------------------------

listening_ports 80

# document_root: top of the web documents repository
document_root /var/www/web
```

This file contains only primary options specific to a given context. They override any same default options possibly defined by the application software.

2.1.2 The secondary parameters

Most secondary configuration parameters are stored in domain H : « Constants and Parameters » as character strings of up to 16 chars. There are « first level » or « master » parameters, which may be immediate (literal) or reference a value or a collection of « second level » parameters. All master parameters are collected in the master parameter list which is stored in contiguous master records in domain H. The first master record is 254 by convention; other records are allocated in descending order. Each record contains 8 fields of 16 chars, field 0 being the check field, which must contain the record number, the other 7 fields containing each one of the following :

- the value of a literal parameter as a char string representing an alphabetic name (like a file name) or a numeric decimal value (or hex if starting with 0x). This value is truncated by the first semi-colon, which allows to append a short comment in the unused space if any. Numeric conversion stops at the first non-numeric digit; the numeric value associated with a non-numeric string is zero. Leading spaces are not permitted.
- the 3-digit number of a record if the parameter stands for
  - a record of secondary parameters also stored in domain H.
  - the base record of a cluster of records in another domain, which is user dependent
- a domain/record/field address (format : Drrrff) for any need to reference a literal value stored in another domain (ex : its length exceeds 16 char). This is transparent : one may change a literal parameter to a referenced one, or conversely, any time, provided that the litteral value cannot be interpreted as an address.

List of master parameters :

index	id	rec	field	name	factory	description
0		254	0		254	(check)
1			1			
2			2	S_TRACE	216	trace masks node
3			3			
4			4	S_SIOPORT	0	COM port number
5			5	S_SIOSPEED	115200	COM line speed
6			6	S_SPIREC	217	first SPI node
7			7	S_ADCREC	228	first ADC node
8		253	0		253	(check)
9	a		1	S_TOFBASE	50	temperature evaluation offset base
10	b		2			
11	c		3	S_LANG	126	list of supported languages
12	d		4	S_MACRO	64	
13	e		5	S_HOME	125	list of home pages
14	f		6	S_SCALING	92	first rec. static values scaling params
15	g		7	S_SCALEFACTOR	101	scalefactor for cumulated analog channels
16	h	252	0		252	(check)
17	i		1	S_GRAPH	236	*subset* graph menus start record in domain O
18	j		2	S_GRAFCHAN	216	*channel* graph menus start record in domain O
19	k		3	S_CHANMENU	196	*channel* menu start record in domain O
20	l		4	S_SEQMENU	216	*sequences* menu start record in domain O
21	m		5	S_HEATING	109	heating / metro
22	n		6	S_DAYLIGHT	128	for sunrise and sunset times
23	o		7	S_DAYSOFF	140	days off calendar

2.2 I/O hardware configuration

2.2.1 General

I/O interfacing may be extended to various devices as long as they feature a hardware/software interface connectable to the computer.

2.2.1.1 Raspberry Pi

The current supported interfaces on the Raspberry Pi are its GPIO pins (except the pins reserved for special functions or protocols) and the interfacing circuits from Microchip supporting the SPI protocol :

- MCP23S17 16-bit I/O expander
- MCP300x and MCP320x series Analogic/Digital Converters (ADC)

A few other peripherals, like a Real Time Clock (RTC) and the Serial Line are directly supported by the Raspbian Operating System (through the I2C protocol and as terminal device `/dev/ttyAMA0`).

2.2.1.2 Serial line

For hosts, like Windows PC, which do not feature a GPIO, the I/O hardware interface is managed by a dedicated microcontroller which communicates with the host via a serial line. A simple character oriented protocol provides the means of interchanging numeric and analogic data.

2.2.2 The MCP23S17 I/O expander

The MCP23S17 I/O expander device provides 16 I/O pins. Up to 8 such devices may be addressed on a single SPI bus, that is 128 pins, but with the current design limit of 64 numeric I/O, only a maximum of 4 devices are supported. The MCP23S17 features two 8-pin **banks**, named A and B, which are treated by software as independent devices. Therefore, if n is the number of devices present in the configuration, numbered from 0 to n-1, the valid bank addresses are in the continuous range 0..(2*n)-1, with an even address for bank A and an odd address for bank B (in other words, if a device/bank address is represented on 3 bits, bits 2 and 1 are the device and bit 0 is the bank). For example, if 2 devices with addresses 0 and 1 are present, the bank address range is 0..3, banks 0 and 1 for device 0, 2 and for device 1. Note that device addresses must be hardware configured continuously, starting at 0;no address gap is permitted

Each pin can be independently configured as input or output by setting an associated bit in in the 'direction' hardware register within the device (0=out, 1=in). In order to support this independence, a small memory database is created in memory and initialized at startup, composed of a collection of configuration *nodes*. Each node contains the information needed for 'bridging' the information flow between the hardware and the application software layer. This information is a table of bytes, each representing a property of the 8 channels of one bank, whose usage as input or output depends on the needs of the application and must be as independent as possible from the channels assignment as seen by the application. Therefore, ideally, this database should allow to associate any I/O pin with any bit in the I/O matrix. To keep things simple and effective, there is one restriction : a bank maps a byte within the I/O matrix pin for bit.

Each node supports from 1 to 8 bits within the same I/O matrix byte and supplies the following information:

Field	Name	Description
0	SPINODID	the SPI node identifier for validity check. This value must be equal to the record number of the node in domain H. If the check fails, the node is assumed to contain inconsistent data and ignored.
1	MAPIDX	the byte index in the I/O matrix (0..7)
2	SPIADDR	the harware I/O address (device and bank address, 0..7)
3	MASKIN	the mask of input pins (1=yes)
4	MASKOUT	the mask of output pins (1=yes) As a pin cannot be both input and output, MASKIN and MASKOUT cannot have any same bits set. This applies collectively for all nodes with the same SPIADDR. The generation of the database is terminated if and as soon as conflicting settings are found. Having no bit set in both MASKIN and MASKOUT is meaningless.
5	PULLUP	the mask of pins with internal pullup resistor (1=yes) This is only meaningful for input pins.
6	POLARITY	the mask of the polarity of I/O pins (1=inverted)
7	-	unused

The record number of the first node is in system parameter S_SPIREC. Subsequent nodes are allocated (and scanned at startup) in ascending record numbers until a node fails to pass the validity check (field SPINODID).
There can be more than one node describing the same bank or associating it to another byte of the I/O matrix, provided this causes no I/O assignment conflict.

2.2.3 The Raspberry Pi I/O pins

The Raspberry Pi I/O pins can be configured simultaneously or/and concurrently with the MCP23S17 I/O expander pins by providing a similar set of nodes, with bank addresses ranging from 8 to 11.
These bank addresses refer to the 32 first pins of the BCM2835 interface and not the Raspberry Pi 40-pin GPIO header. Each bank represents one byte of pins within the BCM2835, in ascending bit order :

Bank	Pin range
8	07-00
9	15-08
10	23-16
11	31-24

Only the pins that are connected to the RPi GPIO header are available :

BCM	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RPi	10	08	33	32	23	19	21	24	26	31	29	7	5	3		
Special	RX	TX			SPI	SPI	SPI	SPI	SPI				I2C	I2C		

BCM	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RPi					13	37	22			15	40	38	35	12	11	36
Special																

If we exclude the SPI and I2C pins, 17 pins are available for GPIO, but all pins must be configured sensibly in the configuration nodes, usually by setting the direction bits to 0 in both MASKIN and MASKOUT for inaccessible or special function pins.

Cross reference table RPi –> BCM

Special	5v	5v	0v	TX	RX		0v			0v		SP	SP	ID	0v		0v			
BCM				14	15	18		23	24		25	8	7			12		16	20	21
RPi	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39
BCM		2	3	4		17	27	22		10	9	11			5	6	13	19	26	
Special	3.3	I2	I2		0v				3.3	SP	SP	SP	0v	ID						0v

3.3 = 3.3V power, 0v = Ground,, SP = SPI, I2 = I2C, ID = reserved for I2C ID EEPROM

2.2.4 The hidden numeric channels

The numeric part of the I/O matrix is internally extended by 8 bytes, allowing to map the GPIO pins to 64 extra numeric channels. These channels are outside the 64-channel numeric addressing space and are therefore *hidden*, i.e. not visible or accessible to the *process(2).* They are mapped by extending the range of the MAPIDX parameter in the configuration nodes from 8 to 15. They cannot have an associated channel table. Their purpose is to fulfil internal ancillary functions, like providing multiplexor addressing, and to keep the visible channels available for application functions.

2.2.5 The MCP3xxx series A/D converters

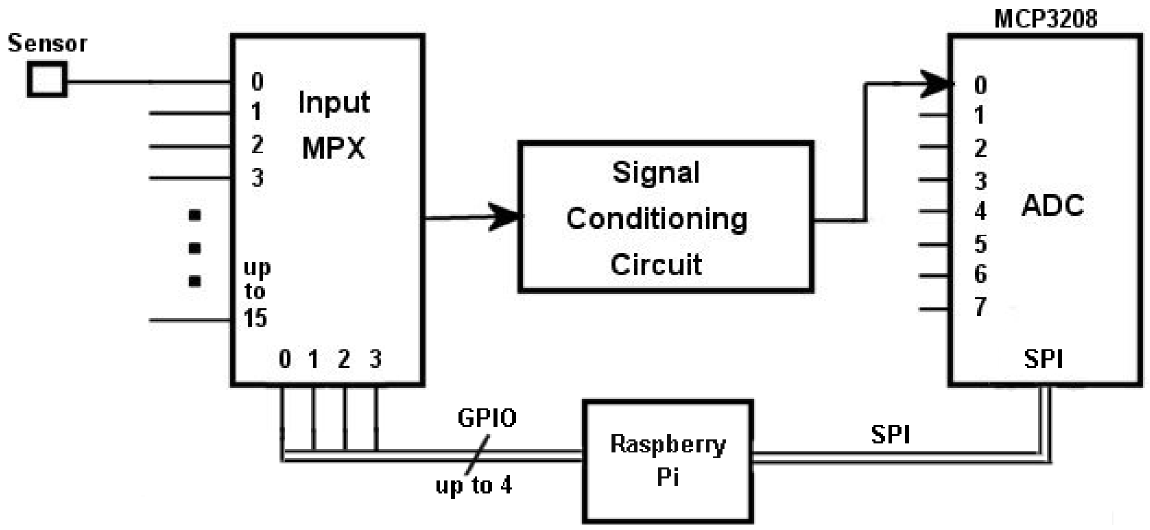
For analog channels, the bridging process is described by **memory objects** which are initialized by the processing of descriptor *nodes* on *disk*. Each object describes from 0 (if empty) to 16 channels that have the same value acquisition method (i.e they represent the same physical value and have some common parameters). Each descriptor node configures a cluster (a contiguous subset) of channels within an object, providing the linking information directing a value got from the hardware to the target software channel (its associated word in the I/O matrix). Nodes are processed in ascending record numbers. If several nodes provide configuration info for the same channel, only the first one is used, later redundant information is ignored.

Information provided by a node :

Field	Name	Description
0	ADCNODID	the ADC node identifier for validity check. This value must be equal to the record number of the node in domain H. If the check fails, the node is considered a terminating node which ends the ADC initialization process.
1	ADCOBJID	ID of the associated object. Number of the used analogic input of the MCP3xxx device (0..7 for the MCP3xx8)
2	ADCFIRST	number of first analog channel in the I/O matrix (64..126)
3	ADCSIZE	number of active channels within object, starting at ADCFIRST within the I/O matrix ADCFIRST+ADCSIZE < 127 ADCMPX+ADCSIZE <= 16
4	ADCDEVICE	ADC device type. Includes signal *rescaling* : 0 reserved for testing 1 10-bit>>2 (MCP30xx to 8-bit) 2 12-bit>>4 (MCP32xx to 8-bit) 3 10-bit raw (MCP30xx) 4 12-bit raw (MCP32xx)
5	ADCDELAY	delay between input mpx address write and data read (W/R), to allow the signal to settle
6	ADCFLIMIT	filter limit (for the filter algorithm)
7	ADCXLIST	list of up to 4 numeric channels switching the hardware multiplexor, coded into a 32-bit value (1 channel per byte). The channel numbers range from 1 to 127, including visible and hidden channels.

The record number of the first node is in system parameter S_ADCREC. Subsequent nodes are allocated (and scanned at startup) in ascending record numbers until a node fails to pass the validity check (field ADCNODID).
As the MCP3xxx series do not support device addressing, the number of supported devices is the number of chip selects available on the RPi for this functionality, a maximum of 2, but only one if the other is used for GPIO expanders.

Input multiplexing
Between the current or temperature sensor and the analogic input of the ADC device, there is a conditioning circuit adapting the sensor signal to the signal type and level expected by the ADC. Up to 16 sensors of the same type may share the same conditioning circuit (which may be bulky and expensive) and ADC channel when they are connected via an input multiplexor. This multiplexor is switched by a group of up to 4 GPIO outputs (either the direct GPIO of the Raspberry Pi, or expanded outputs) making up a multiplexor address. Up to 8 of such groups may be defined, one for each of the used inputs of the ADC device. This depends on the characteristics of the different conditioning circuits : they usually differ by the scanning frequency and the time for the signal to settle after a switching, that is the delay between input select and input read (current signals should be scanned



as frequently as possible but as the conditioning circuit includes a rectifier and filter, 50 or 60 Hz RMS signals may take a second or two to settle; temperature signals are DC signals that vary very slowly, there is no need to hurry in scanning them, but as their conditioning circuit includes only DC amplifiers, the time to settle may be very short).

RPi direct GPIO pins available for MPX addressing :

		MPX address bits			
	Bank	*3*	*2*	*1*	*0*
GPIO pins	10	**19**	**18**	**17**	**16**
RPi pins		*35*	*12*	*11*	*36*
GPIO pins	10	**23**	**22**	**21**	**20**
RPi pins		*16*	*15*	*40*	*38*
GPIO pins	11	**27**	**26**	**25**	**24**
RPi pins		*13*	*37*	*22*	*18*

All pins used for MPX addressing must be configured. It is recommended that all pins of a group belong to the same bank, as a configuration node is required for each used bank. The above selection fulfils this condition.

Input filter algorithm
In a home environment, most analogic signals vary from moderately slowly (electric consumption, except for switching on and off) to very slowly (temperature). It can therefore be assumed that small fast signal variations are random perturbations or jitter that should be filtered out.
The following algorithm is provided. Let :

NEW	the new value provided by the hardware
new	the filtered NEW passed to the application
old	the previous *new*
delta	the signed difference between *NEW* and *old*
sum(delta)	*delta* cumulated
flimit	a value provided by the user, delimiting the 'small' delta range below 0 disables filtering

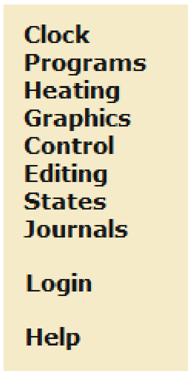
If *delta* is :

- large : one is twice the other or more *: new* is updated immediately to *NEW*
- medium : larger than the user programmed *flimit* : *new* becomes *(NEW+old)/2*
- small : smaller than the user programmed *flimit* : *delta* is cumulated (summed up). When the absolute value of *sum(delta)* exceeds *flimit*, *new* is modified by -1 or +1 and *sum(delta)* is cleared.

2.2.6 Serial line I/O
Currently not supported on the RPi.

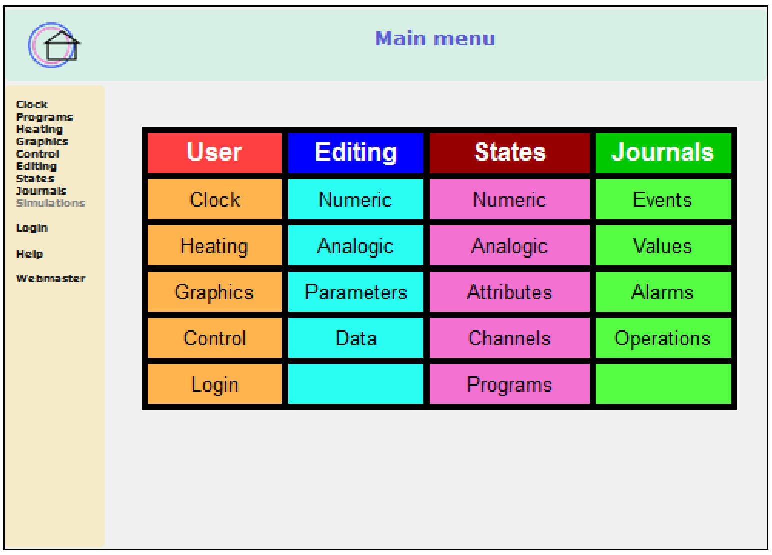
3 The user interface

3.1 Main menu



The user interface is made of a collection of dynamic web pages, featuring all the same alphabetic first level menu in their left column. Clicking on one of the entries of this menu either scrolls down a second level menu immediately below the entry or loads another page if there is no second level menu. This menu makes all possible options available within one or two clicks.

The main menu page, which acts as a home page, provides a one click access to the most frequent options.



User	Editing	States	Journals
Clock	Numeric	Numeric	Events
Heating	Analogic	Analogic	Values
Graphics	Parameters	Attributes	Alarms
Control	Data	Channels	Operations
Login		Programs	

3.2 Login

User privileges

Each user is assigned a level of privilege, associated with one of the four following categories :

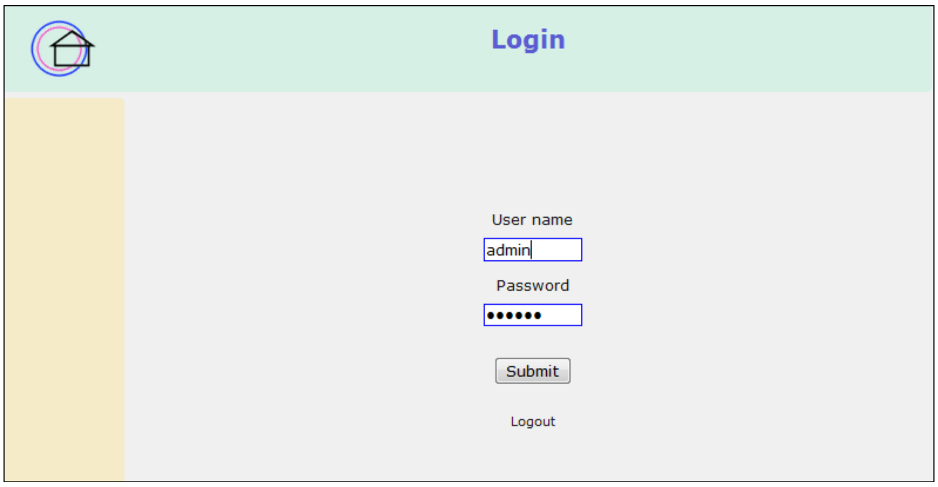
Visitor	no privilege, no login required, has read-only access to a subset of the data
Read access	can view data that is hidden to visitors, but cannot modify it
Write access	may modify data, except if reserved to the administrator
Administrator	full access

Privileged users must login to activate their privileges. The system supports up to 7 privileged users. Only the administrator has access to the user authorization domain, which is not visible to any other user, and can create or delete any user and set privileges (including self, beware !). There should be only one user with administrator privileges, who must be on top of the user's list (user number 1).

Each user transaction requires its own level of privilege and is otherwise rejected with the message « Privileged operation ». Write or administrator access is required for any modification of the database.

Login procedure

This page has no left column menu, as it is opened on top of the calling page to which control returns after the login is completed and the page closes.



When clicking on the **Submit** button after having entered a valid username/password combination, the long name of the user is displayed and the caption of the Submit button changes to **Exit**. Clicking on the Exit button validates the new user and the page closes.
If the username/password combination is invalid, the message « Unknown user » is displayed. After 3 failures in a row, the login operation is given up.

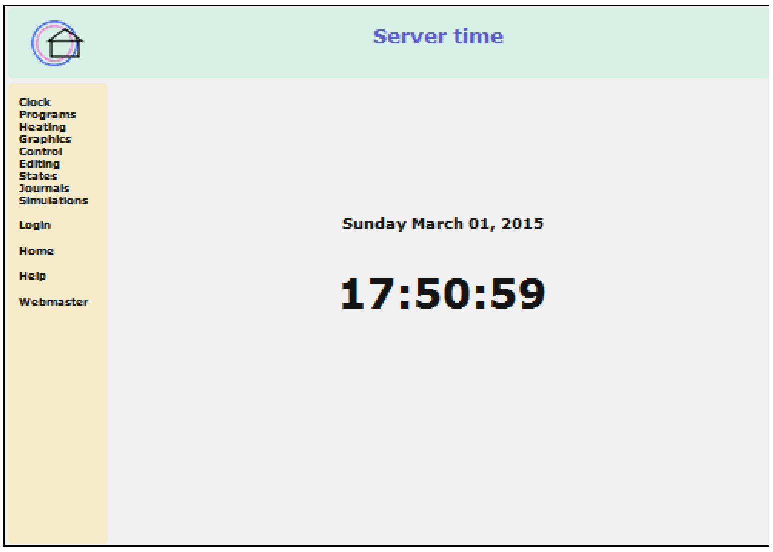
Logout

The Login page is also used for logging out. In fact, when the login page is loaded, it immediately logs out the current user, meaning that the user must perform a successful login to recover their access rights. Clicking on the Submit button with both fields blank completes Logout.
Some menu entries may appear grayed out, or even hidden. To make them active, the user must reload the calling page after login, load another page or hit CTRL/F5.
A login is only valid for the current browser session. Exiting the browser terminates the login.
An automatic logout may occur if a session times out.

Disabled login

For secure or development environments, it is possible to have permanent administrator privileges without login by setting bit 0 in the OPTMASK option in the *configuration file*.

3.3 Clock



The server uses its own local time for all its internal needs like managing timed events, delays and timestamps for journal entries. This time may differ from the user's time if no time synchronization mechanism exists and obviously if the user is not in the same time zone.
The display format of the date may be language dependent.
The time is shown with seconds. On heavy server or network load, the updating may occasionally miss a second or two.

3.4 Data edition

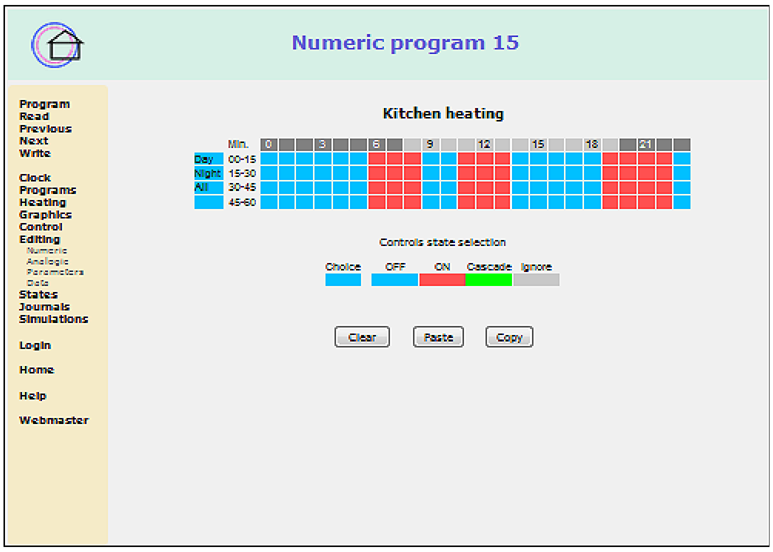
The database is maintained with several editing interfaces depending on the type of data to be edited.
Formatted editing is used for :

- programs, i.e. collections of 96 binary values are binary stored in byte records : *numeric and analogic program editors,* with a click-on-value interface
- channel tables, whose parameters must be easily idenfiable : *numeric and analogic channel tables editors (menu entry :* **Parameters**). There, numeric values must be entered.

Unformatted editing is used for everything else:

- raw text fields with no specific identification : *catch-all editor (menu entry:* Data*)*

3.4.1 Numeric programs



A **numeric program** is a time-of-day ordered list of values that are used to set numeric outputs or to compare with numeric inputs all along a day. This function is modelled on the standard electro-mechanical timer shown left.

The standard functionality features 96 values, each valid during a *quarter hour* (15 minutes). Any program may be associated with any numeric channel to make up a timer (output) or an event monitor (input).

This page allows to program the 96 values individually or bulk (the same value for all, or for day or night, or for a whole hour).

To display a page :

- open the **Editing** submenu and click on the *Numeric* entry. Then enter the program number (1..255) in the topmost white field and hit the Enter key twice.
- alternately, a menu allows to select a program by its name within a list up to 20 programs : open the **Programs** submenu and select the *Numeric* entry, then click on the wanted program name (this list is made up by the system administrator)

To program a value :

- first select this value on the selection bar (this value is reflected in the choice box and in the leftmost column),
- then click on the appropriate quarter hour box, or, for bulk programming, on the hour caption bar or one of the boxes of the leftmost column.

In addition to ON-OFF, the following values may be selected:

- *Cascade:* the value is got from another source
- *Ignore:* no automatic check or action is performed by the system : the channel is left alone.

The 3 buttons allow to *Clear* a program (set all values to *Ignore*) or to make a temporary *Copy* of the program before a modification in order to restore it *(Paste)* to cancel this modification. This is also usable to duplicate a program.
The blinking box denotes the current server time.

The action menu on the top of the left column provides the following functions :

Program	selection of a new program by entering its number in the top most box
Read	refreshes the grid of the current program
Previous	displays the previous program
Next	displays the next program

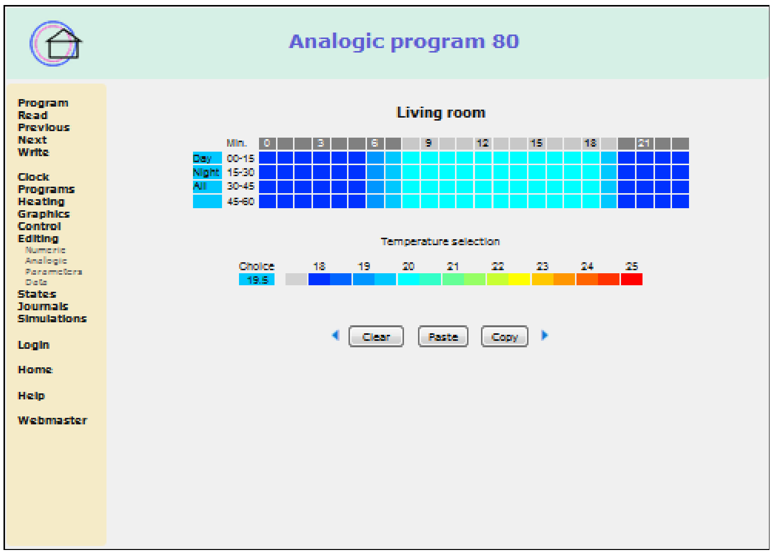
Write writes the current program to disk, overwriting the old one (privileges required). Failing to **Write** causes the loss of any modification since the previous Write

There are up to 255 possible numeric programs. In the range 128-255, an automatic offset is added to the program number in two cases :

- if the program number is the one specified in system parameter S_DAYLIGHT, the offset is the number of the month (january=0, december=11). This allows to make a lighting control time-of-year dependent. If this feature is used, the 12 programs should be reserved for it.
- otherwise, the offset is day-of-week dependent (work days=0, Saturdays=1, Sundays=2, special days=3, see **special days** below)

3.4.2 Analogic programs

An **Analogic program** is a time-of-day ordered list of values that are used as successive values of an analogic setpoint all along a day.
The standard functionality features 96 values, each valid during a *quarter hour* (15 minutes). Any program may be associated with any analogic channel to provide a setpoint to an external device or to a regulation loop.
This page allows to program the 96 values individually or bulk (the same value for all, or for day or night, or for a whole hour).



To display a page :

- open the **Editing** submenu and click on the *Analogic* entry. Then enter the program number (1..255) in the topmost white field and hit the Enter key twice.
- alternately, you may select a program by its name within a list up to 20 programs : open the **Programs** submenu and select the *Analogic* entry, then click on the wanted program name.

To program a value :

- first select this value as a color on the selection bar (this value is reflected in the choice box and in the leftmost column),
- then click on the appropriate quarter hour box, or, for bulk programming, on the hour bar or one of the boxes of the leftmost column.

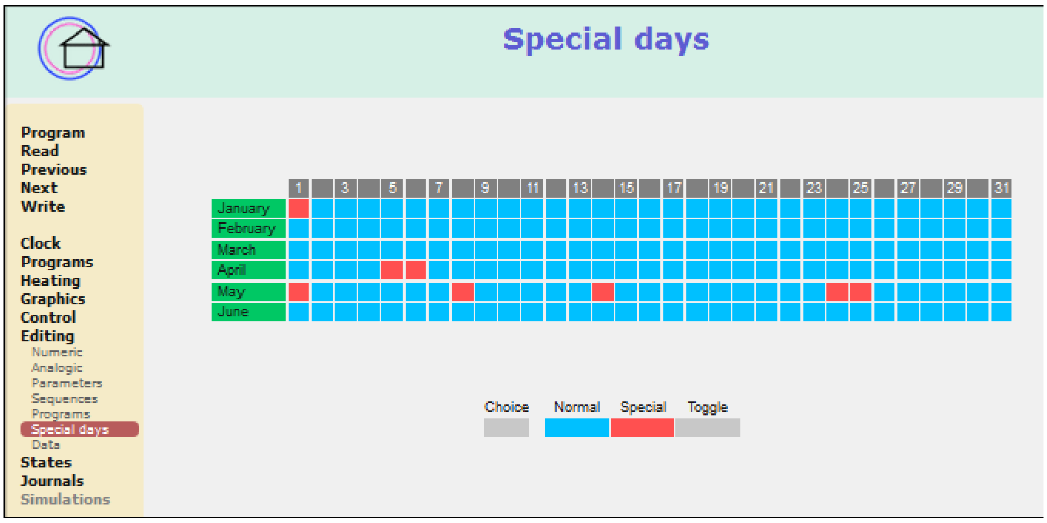
In this example, it is possible to choose a temperature comprised between 17.5 and 25.0 °C by half degree increments. This is the default range and should be appropriate for a temperature regulation of a living or working space. Other temperature ranges are available by clicking on the arrow heads left (colder) or right (warmer) of the buttons : the temperature captions line changes.
If a setpoint has been programmed within a given temperature range, it cannot be displayed correctly in another range : its color is replaced with a shade of gray : light gray if the setting is above the range (increase the range to catch up), dark gray if the setting is below the range (decrease the range to catch up). The initial range at page load is selected by the setting of the first box (00:00).

The 3 buttons allow to *Clear* a program (set all values to the minimum of the current range), or to make a temporary *Copy* of the program before a modification in order to restore it *(Paste)* to cancel this modification. This is also usable to duplicate a program.
The blinking box denotes the current server time.
The top-left action menu operates exactly as for numeric programs.

There are up to 255 possible analogic programs. In the range 128-255, an automatic offset is added to the program number as for numeric programs.

3.4.3 Special days

They are days that are different from normal working days, Saturdays or Sundays for whatever reason (days off, celebrations, vacations, holidays...). They are user programmable for one year (the rest of the current year and the beginning of the next year) on two semester pages. Just click on a day's box after selecting the appropriate color. Clicking on the choice field clears the whole semester.



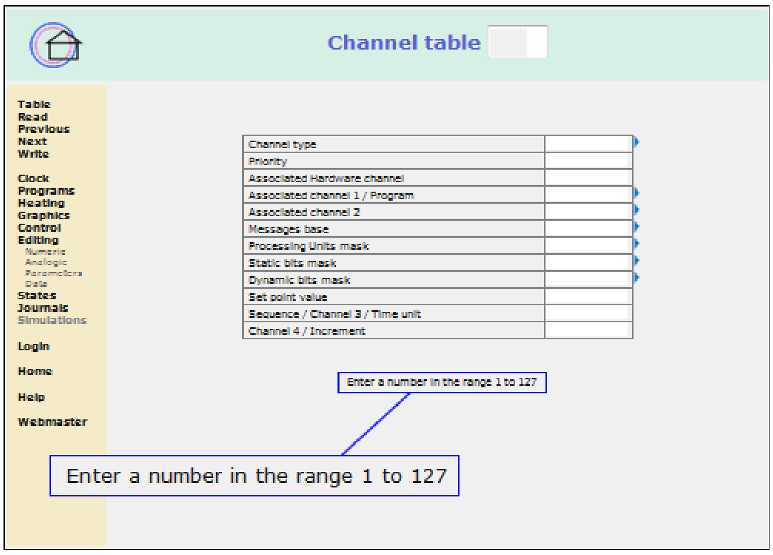
If a program number is larger than 127, an offset (0=working day, 1=Saturday, 2=Sunday, 3=special day) is added to it before use. The special day attribute overrides any other attribute (a Saturday or Sunday becomes a special day if marked as such).

3.4.4 Numeric channel tables

This interface provides the editing capability of all parameters associated with any of the 63 possible numeric channels (numbered 1 to 63). These parameters are all numeric decimal values, stored as character strings in the database on disk in the domain E.

To display a page, use the following procedure :

- open the **Editing** submenu and click on the *Parameters* entry. The following empty page is displayed :



Channel type	
Priority	
Associated Hardware channel	
Associated channel 1 / Program	
Associated channel 2	
Messages base	
Processing Units mask	
Static bits mask	
Dynamic bits mask	
Set point value	
Sequence / Channel 3 / Time unit	
Channel 4 / Increment	

Enter a number in the range 1 to 127

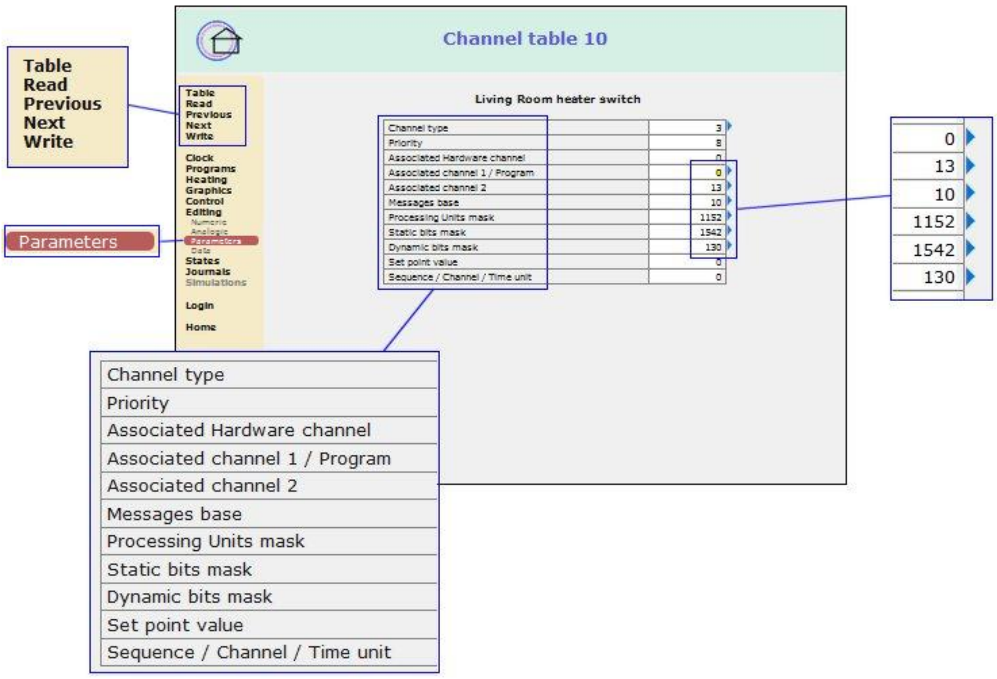
Enter a number in the range 1 to 127

- Enter a channel number *(1 to 63 for numeric channels)* in the topmost white field and hit the Enter key twice. This fills the right column of the grid with the current values of the parameters of the selected channel.

  After the first hit on Enter, the following status message is displayed :

  Read : get the new record. **Write** : write the current record to disk

  This allows to copy the currently displayed values to the selected channel, providing a channel table duplication function. The second Enter skips this function.



Channel table 10

Living Room heater switch

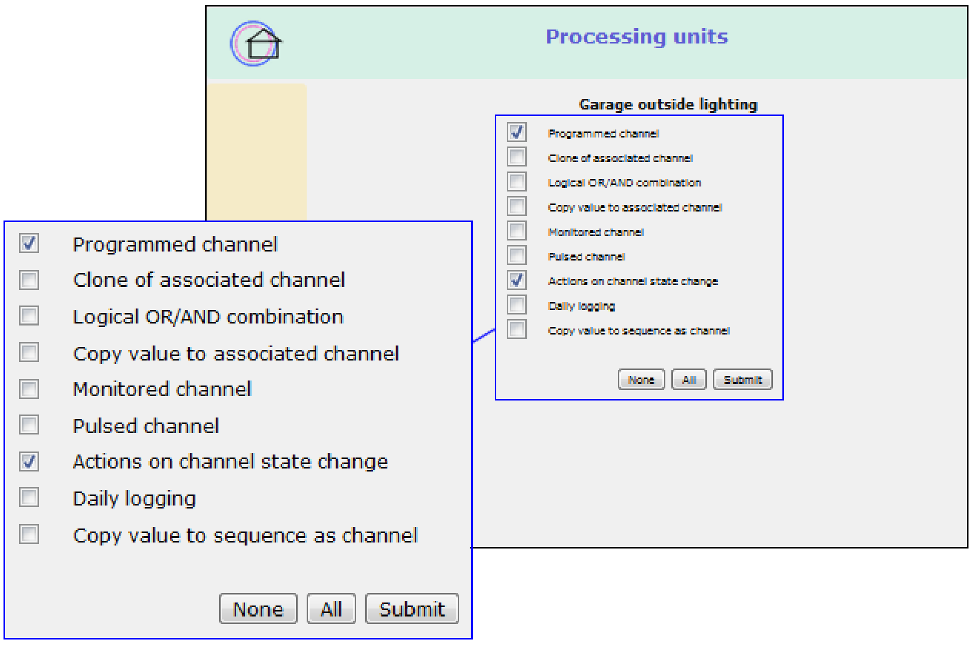
Channel type	3
Priority	8
Associated Hardware channel	0
Associated channel 1 / Program	0
Associated channel 2	13
Messages base	10
Processing Units mask	1152
Static bits mask	1542
Dynamic bits mask	130
Set point value	0
Sequence / Channel / Time unit	0

- Edit any value in one of the fields in the right column. Do not hit Enter, use the Tab key or the mouse to change fields.
- When ready to save the new values, click on **Write** in the top left action menu. This makes the new values permanent. Failing to do this action before proceeding with another operation causes the modifications to be lost. Clicking on **Read** restores the displayed values to their state after the last successful Write operation.
- watch the space below the table for any prompt/status (in black) or warning/error (in red) message. The « Privileged operation » message indicates that you did not **Login** or your user privilege level is not high enough and your action was rejected.

The action menu on the top of the left column provides the following functions :

Table	selection of a new channel by entering its number in the topmost box
Read	refreshes the parameter values of the current channel
Previous	displays the parameter values of the previous channel
Next	displays the parameter values of the next channel
Write	writes the parameter values of the current channel to disk

Many of the parameter values represent the encoded selection of a collection of attributes of the channel, which would be cumbersome for the user to be handled numerically. To make it more user friendly, a detailed clear text list of attributes is displayed by clicking on the blue arrowhead at the right of a field, for example for the « Processing Units mask » :


Processing units
Garage outside lighting
☑ Programmed channel
☐ Clone of associated channel
☐ Logical OR/AND combination
☐ Copy value to associated channel
☐ Monitored channel
☐ Pulsed channel
☑ Actions on channel state change
☐ Daily logging
☐ Copy value to sequence as channel
None
All
Submit

Functions of the buttons :

Check boxes	individual selection of one or several items as attributes of the channel
None	deselects all checkboxes
All	selects all checkboxes
Submit	copy the checked boxes mask value to the calling page and exit
✕	cancel and exit (browser button)

This page has no left-column menu as it *must* exit to the calling page.
There is no attributes incompatibility detection. Correct selection is under the user's responsibility.

3.4.5 Analogic channel tables

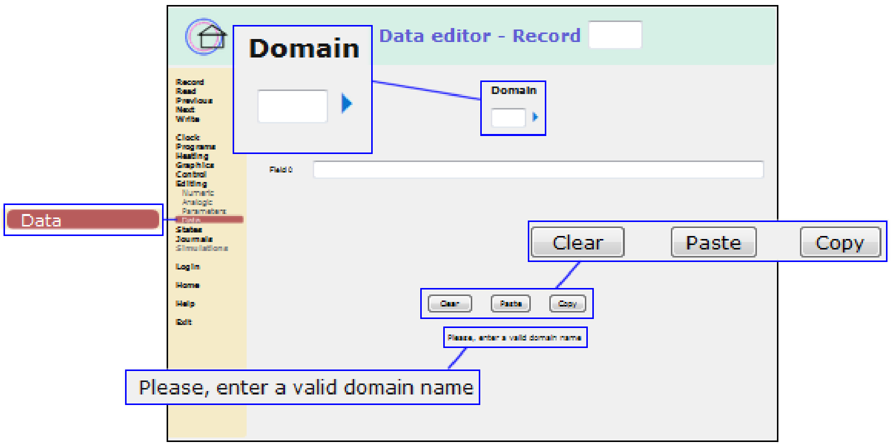
The procedure is the same as for numeric channel tables, except that the channel number ranges from 64 to 127 and the parameter list is slightly modified and extended.

3.4.6 The catch-all editor

This is the general low level maintenance tool of the database, used in all cases not handled by any of the previous editors. It considers the database as a collection of independent domains and reformats its interface according to the field structure of the domain, providing an input field for each of the 8 first data fields within the record currently edited. All fields are alphanumeric. There is no field captions other than Field 0 ... Field 7 as the field contents depend on domain and usage.
A few domains and fields beyond the eighth are not editable. Domains editable with other editors should not be modified with the catch-all editor.

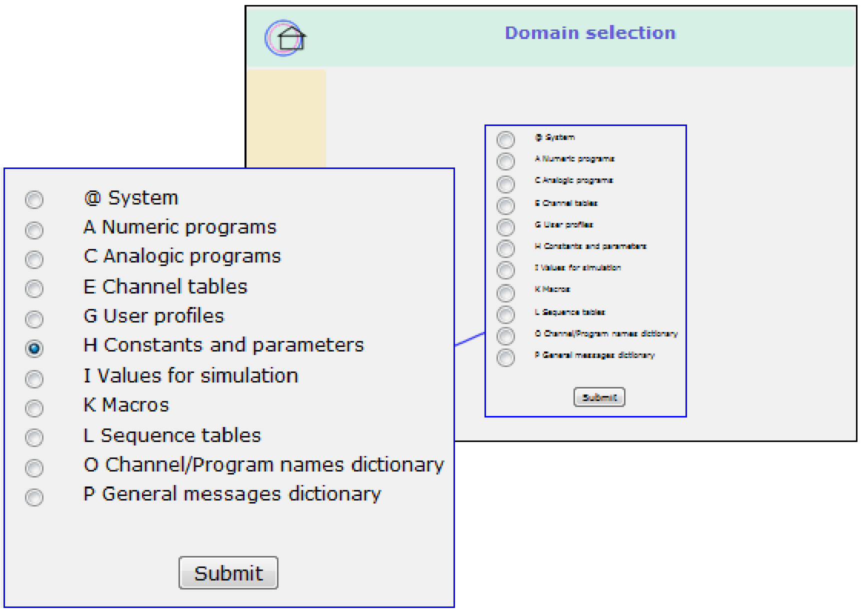
Editiong session

- click on the Data item in the **Editing** submenu. The empty editor page is displayed



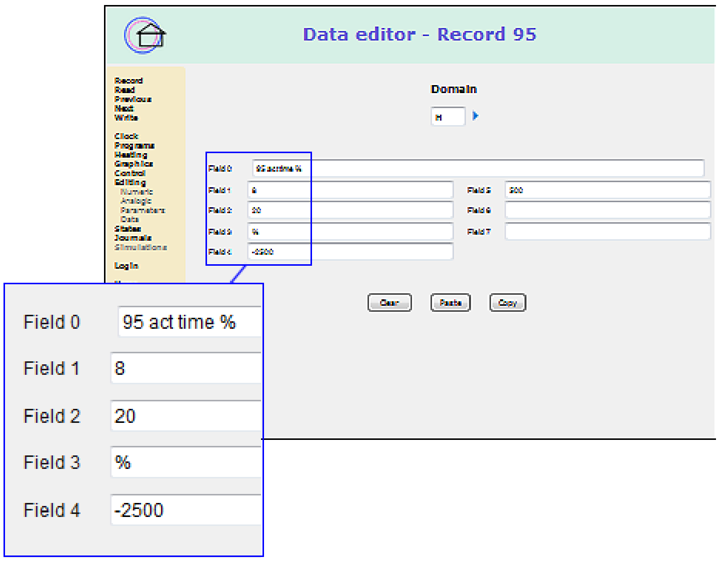
Please, enter a valid domain name

- enter the letter of the domain to edit or click on the blue arrow head on the right of the domain field for opening a domain selection page :



- ○ @ System
- ○ A Numeric programs
- ○ C Analogic programs
- ○ E Channel tables
- ○ G User profiles
- ● H Constants and parameters
- ○ I Values for simulation
- ○ K Macros
- ○ L Sequence tables
- ○ O Channel/Program names dictionary
- ○ P General messages dictionary

- click on the radio button of the domain to select and Submit. The selection page exits, the letter is loaded into the domain field and up to 8 empty edit boxes appear.
- enter the record number and hit the Enter key twice. The boxes are filled with the contents of the fields of the selected record.



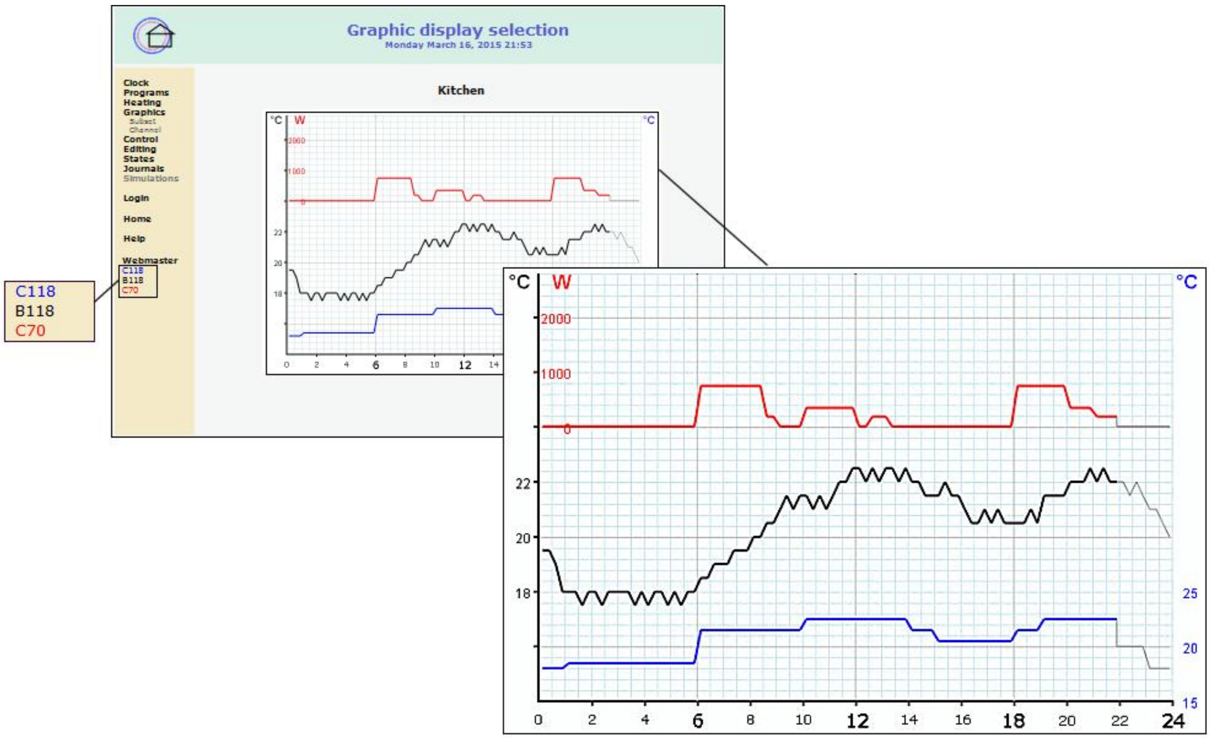
Domain: H

Field 0	95 act time %
Field 1	8
Field 2	20
Field 3	%
Field 4	-2500
Field 5	500
Field 6	
Field 7	

- for editing operations and saving the modifications, do as with the channel table editor.

3.5 Graphics

This feature displays up to 4 curves, each with its own vertical graduation, on the same chart. It uses the daily logging function and its collection of 96 values for each channel which has this functionality enabled as processing unit and in its static bits mask. A list of up to 4 channels is appended as argument to the page URL and the corresponding daily values are displayed as curves with each its own colour, which is also the colour of the graduation and of its channel number which appears on a clickable list in the left column. Clicking on one of these numbers displays the table page of the channel.



The curves represent the evolution of today’s physical values from 00h00 to now (on the example above : 22h00). The parts of the curves that follow ‘now’ are yesterday’s values, in light gray.
There is one value for every quarter hour (15 minutes interval, 96 values for a whole day). For technical reasons, there is no time alignment at startup, meaning that the timing on the chart may be shifted by up to one value. The chart is automatically refreshed every half-quarter.

3.5.1 The channels list

The list appended to the page URL is composed of a hash character (#) followed by up to 5 channel identifiers separated by underscores. Example :

```
…/celsiusA.htm#A216_b118_C118_C70
```

A channel identifier is an alphabetic source letter followed by a channel number. The source letter indicates the origin of the data :

A	Chart caption. The channel number is the index of the entry in domain O
B	Setpoint. The channel number is the index of the entry in domain A
C	Daily logging values. Nothing if the channe lis not correctly configured.
D	Simulation values. The channel number is the index of the entry in domain I
E	Same as C
F-Z	Ignored

As the same channel may provide data from different sources, it may appear more than once in the list, with different source letters, but only two occurences are fully supported, one with an upper case source letter, the other with a lower case source letter.

3.5.2 Display configuration

Each curve and its graduation are configured by a set of 8 parameters contained in a record of the system parameter domain H. The *Channel type* entry in the *channel table* is a relative index within a cluster of such records (which starts at the record number in system parameter S_SCALING). The first 4 records have names suggesting their expected use :

Index	Name
0	(Reserved)
1	Temperature
2	Numeric variable
3	Percent
4	Electric power

But these and all the next ones can be freely reprogrammed.
A *lower case* source letter adds 8 to the *Channel type.* For example, if channel 70 has a channel type of 1, the channel type would be 1 for B70 and 9 for b70. This must be carefully considered when

allocating channel type numbers and records to avoid collisions. Otherwise, the source letter is case insensitive.

3.5.3 Configuration parameters

Field	Contents	Example
0	channel plotting type	1
1	slope	5
2	offset	225
3	physical unit name or symbol	°C
4	graduation range min	-225
5	graduation range max	300
6	horizontal position of vertical scale	1
7	visible part of the graduation	70

channel plotting type

1 Slant - a point is joined to the next by one slanting segment
2 Step - a point is joined to the next by a vertical segment followed by a horizontal segment

slope / offset

As the raw physical values are stored in bytes, these 2 parameters are used to rescale these values according to the formula :

```
displayed_value = (slope * raw_value) – offset
```

Note the minus sign before the `offset`.

For a representation as integers, the values entered for these parameters must be magnified 10 times.

physical unit name or symbol

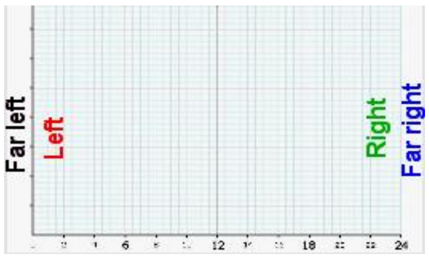
Any unit name like °C % kW m^3 , etc. or blank for no display.
The symbol is displayed on top of the grid, regardless of the vertical position of the graduation.

graduation ranges min / max

The software uses these values to select a graduation that is easy to read and reevaluates the displayed values accordingly. Negative as well as positive values are possible.
The values entered for these parametes must be magnified 10 times.

horizontal position of vertical graduation

0 default = channel plotting type
1 far left
2 left
3 right
4 far right
5 left enlarged
6 far right enlarged



visible part of the graduation

A double digit decimal number :
units : lowest visible value (0..7)
tens : highest visible value (>= lowest)

The vertical axis can receive up to 8 equid 7, bottom-up. Use this feature if only a few of them should be visible or to avoid overwriting.

Example (as used with the diagram above) :

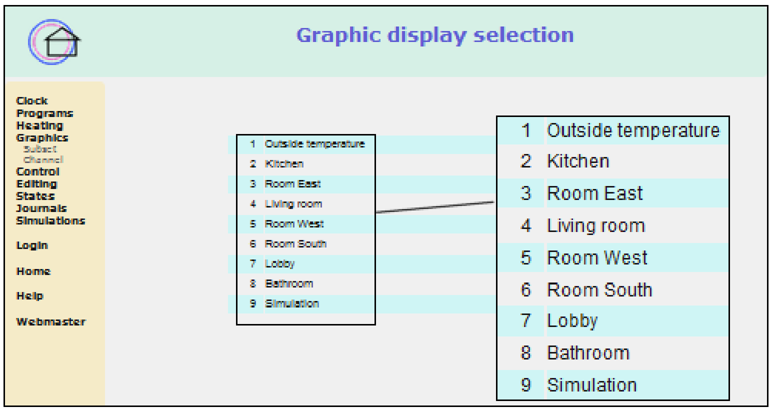
Channel Id	b118	C118	C70
Type	1	1	4
Rec. index	9	1	4
Field 0	1	1	1
1	5	5	100
2	225	225	0
3	°C	°C	W
4	160	150	-50000
5	500	250	30000
6	4	1	5
7	20	42	75

3.5.4 Named lists of channels

Two user programmable sets of named lists are available as clickable menus :

Subset : display a collection of related channels

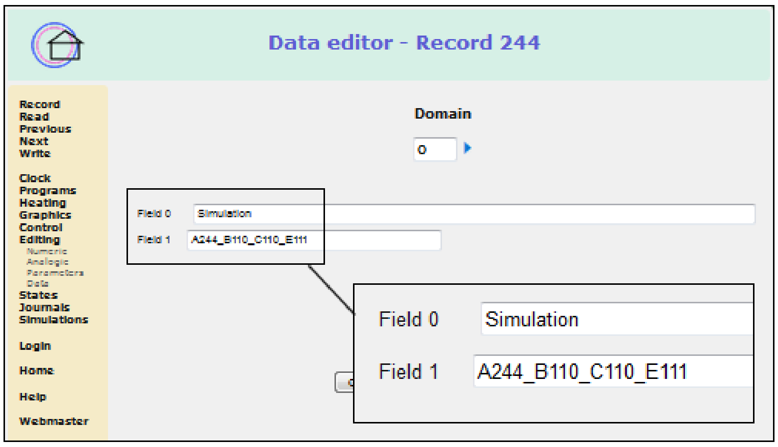
Channel : display one channel



1	Outside temperature
2	Kitchen
3	Room East
4	Living room
5	Room West
6	Room South
7	Lobby
8	Bathroom
9	Simulation

There are 2 sets for commodity, but in fact they are identical in operation, differing only by their names in the main menu and the number of their base records in domain O (system parameters S_GRAPH and S_GRAFCHAN).

To add an entry to one of the 2 sets, use the catch-all editor, open domain O and select the first blank record at the end of the set. then enter the name of the new list in field 0 and the channel list (without the leading hash character) in field 1. (A record is blank if its first character in field 0 is a space).

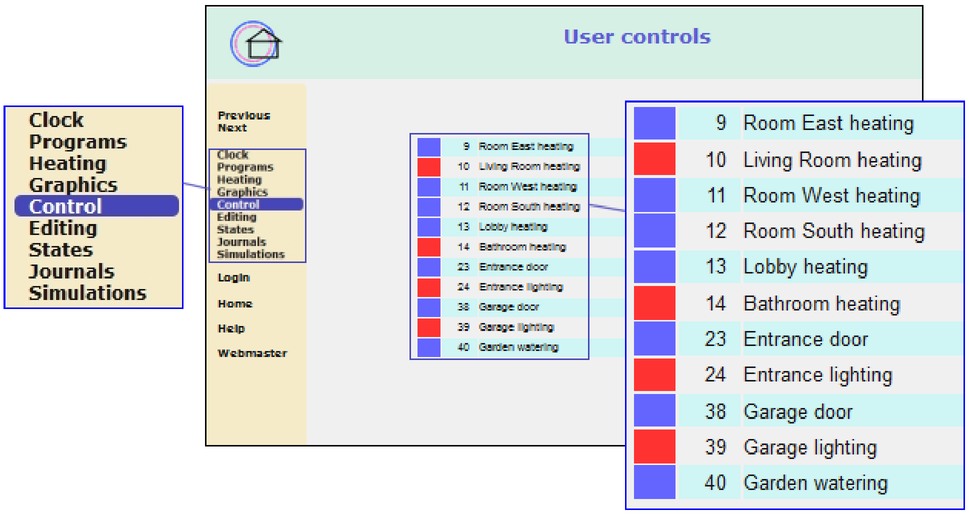


Field 0	Simulation
Field 1	A244_B110_C110_E111

Of course, it is always possible to modify a list any time, but you cannot remove a list without closing the gap, as a blank record acts as a set terminator.

3.6 User controls

This page displays a list of numeric output channels that can be directly set or reset by the user. The colour of the box shows the current state of the channel : blue=OFF, red=ON. Clicking on a box toggles the state of the channel. Clicking on the channel name opens the channel table page.



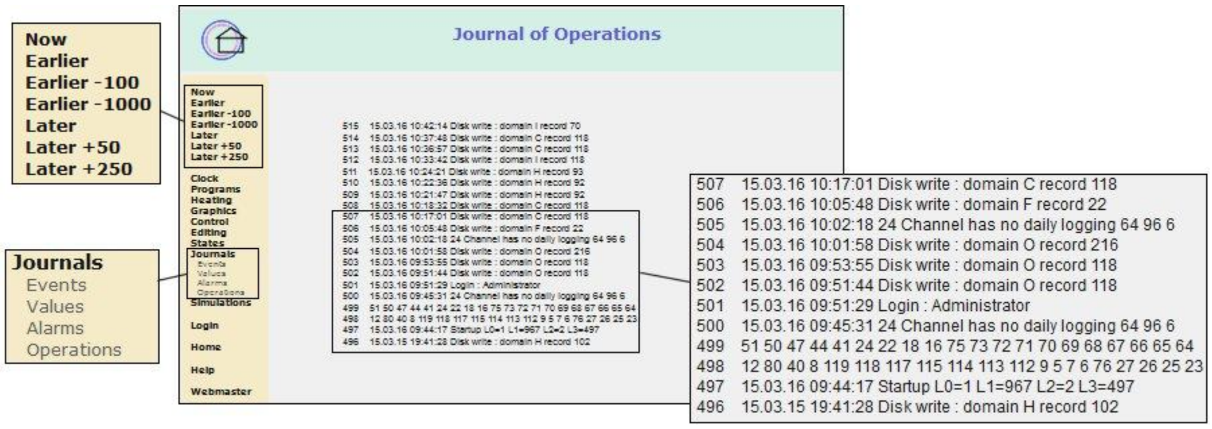
🟦	9	Room East heating
🟥	10	Living Room heating
🟦	11	Room West heating
🟦	12	Room South heating
🟦	13	Lobby heating
🟥	14	Bathroom heating
🟦	23	Entrance door
🟥	24	Entrance lighting
🟦	38	Garage door
🟥	39	Garage lighting
🟦	40	Garden watering

The contents of the list is user defined by checking the box « User control » in the entry « Static bits mask » of the channel table. It is the user's responsibility to evaluate if this function would conflict with other functions configured on the channel.

The list may contain up to 32 channels ordered by increasing channel numbers, in two pages (commands **Previous** and **Next**).

3.7 Journalling

The system generates or witnesses *events*, which it has the possibility to record in one or several of 4 journalling domains. The logged information can be accessed any time by the user by calling one of 4 journal pages.



```
515   15.03.16 10:42:14 Disk write : domain I record 70
514   15.03.16 10:37:48 Disk write : domain C record 118
513   15.03.16 10:36:57 Disk write : domain C record 118
512   15.03.16 10:33:42 Disk write : domain I record 118
511   15.03.16 10:24:21 Disk write : domain H record 93
510   15.03.16 10:22:36 Disk write : domain H record 92
509   15.03.16 10:21:47 Disk write : domain H record 92
508   15.03.16 10:18:32 Disk write : domain C record 118
507   15.03.16 10:17:01 Disk write : domain C record 118
506   15.03.16 10:05:48 Disk write : domain F record 22
505   15.03.16 10:02:18 24 Channel has no daily logging 64 96 6
504   15.03.16 10:01:58 Disk write : domain O record 216
503   15.03.16 09:53:55 Disk write : domain O record 118
502   15.03.16 09:51:44 Disk write : domain O record 118
501   15.03.16 09:51:29 Login : Administrator
500   15.03.16 09:45:31 24 Channel has no daily logging 64 96 6
499   51 50 47 44 41 24 22 18 16 75 73 72 71 70 69 68 67 66 65 64
498   12 80 40 8 119 118 117 115 114 113 112 9 5 7 6 76 27 26 25 23
497   15.03.16 09:44:17 Startup L0=1 L1=967 L2=2 L3=497
496   15.03.15 19:41:28 Disk write : domain H record 102
```

An event is a significant change in the state of the system, for example :

- system startup
- the change of state of a numeric channel
- the crossing of a threshold of an analogic channel
- the action of an operator (login or modification of the database)
- the change of date for channels with daily logging

For commodity, the 4 journals are named *Events, Values, Alarms, Operations*. This provides a rough sorting by event type, but imposes no restriction to which journals an event is directed to, except that the Operations journal is only accessible to the system administrator.

3.7.1 Viewing the contents of a journalling domain

When a journal page loads, it displays the 20 most recent messages in reverse chronological order. Click on **Earlier** to get the 20 immediately older messages and on **Later** to get the 20 immediately more recent messages (if possible). As a journalling domain can contain up to one million entries (depending on the size of the associated file), commands are provided for moving the message index back and forth by several hundreds of messages. Ckick on **Now** to get back to the most recent messages.

3.7.2 Managing the message library

Each recordable event is associated with a number, which is the record number of its associated text in domain P.

- system message numbers are hard coded and allocated in decreasing values, starting at 255
- the number associated with a channel is in entry *Message base* of its channel table. If several messages are needed for the same channel, they are collected into a message cluster allocated upwards and starting at *Message base*.

Each message in domain P is in fact a message format string including formatting directives, which are percent characters followed by one letter representing an action on an associated argument (similar to the *printf()* function of the C standard library). The server parses the string and appends to it the string argument corresponding to each directive found and sends the complete string to the *client(2),* which formats and displays it. The arguments are separated by carets (^). Percent signs and carets must not appear alone in the format string, they must be escaped by a percent sign. In a few cases, where the resulting length of the string would cause overflow problems, the original format string is itself patched.

The system administrator has the responsibility to populate the domain P with format strings with the correct syntax.

List of available formatting commands :

Specific means that the associated argument is not supplied by the system (like date and time) and is dependent on the context in which the message logging call is performed. These letters require the knowledge of the ordered list of available specific arguments and should be used very carefully.

Letter	Action	Specific
^	escaped caret. Passed as an argument containing '&#94 ;'	
%	escaped percent. Passed as an argument containing '%'	
d	argument converted to decimal string	yes
H	current system date as YYMMDD	
h	current system time as hh :mm :ss	
i	argument converted to decimal string	yes
K	must be followed by 4 dummy characters : %K…. The six chars are overwritten with the system date YYMMDD	
n	number of the used format string	
N	no action, does not generate an appended argument. Provided for allowing the *client(2)* to insert a local argument	
p	tells the client to skip the associated argument	yes
s	special. Used to pass ^ and % as arguments. Do not use.	
u	must be followed by a double digit decimal number (possibly with a leading 0) and associated with a numeric argument. The sum of this number and the argument is the index into the *state words table* (supplied by the client). %udd is then replaced on the client side by the indexed state word *(off, on, open, closed, low, high...)*	yes
Other	Reserved, must not be used	

Example : Assume 2 specific arguments with the values 7 and 31 and the following format string at number 61 :

```
1%H%h %n Room %i hygrometry %d%%
```

Before storing it in Journal 0, the server would convert it to :

```
1%H%h %n Room %i hygrometry %d%s^150317^13:47^61^7^31^%^
```

The client could reformat and display it as :

```
17/03/15 13:47 61 Room 7 hygrometry 31%
```

Journal files selector mask

The first char of a message text may have a special meaning, allowing to select from 0 to 4 journal files to receive the entry. If this char is :

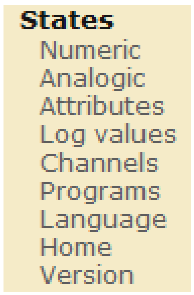
- '0', this message is ignored (used for testing or for disabling a given message)
- in the range '1'..'?', the four low order bits of its code are used as a mask of the journal files to receive the entry
- else, the message is sent to journal 0

Standard specific arguments

Numeric channels
- channel number
- channel state : OFF=0, ON=1

Analogic channels
- channel number
- channel transition : normal->low=0, low->normal=1, high->normal=2, normal->high=3
- channel full setpoint (including the additional temporary increment/decrement)

3.8 States



The **States** submenu provides a number of functions mostly dedicated to maintenance and development.

Numeric : A table of the current state of the numeric channels 1 to 59 (channels 60 to 63 are not visible on this chart). Blue=OFF, red=ON. You can change the state of a channel by clicking on its box. The new state depends on the selected action by clicking on the selection bar (OFF ON or Toggle). Of course, clicking on an input channel or an output channel directly controlled by the system is not meaningful and may cause unexpected effects. The table is refreshed every 10 seconds with the true state of the channels.

Analogic : A table of the current *raw values* of the analogic channels. Green boxes denote channels that have a channel table containing a *physical value*, which can be displayed for a few seconds by clicking on the box.

Attributes : A table displaying for each channel a bit pattern representing 5 of its attributes. Bits are numbered from right to left and stand for the following attributes :

Bit	Attribute
4	Channel has a daily logging
3	Channel has a channel table
2	Channel with user function
1	Channel write locked
0	Channel numeric state

Log values : By selecting a channel by its number in the top most box, its current 96 daily logging *physical* values are displayed, provided this channel has a channel table and this function enabled.

Channels : A list with on each line the channel number followed by the name assigned to the channel or blank if no assignment. This provides a synopsis of all assignments and an aid to channel allocation during application development. Click on **Previous** or **Next** to display a new set of 20 channels.

Programs : As for **Channels**, a list with on each line the program number followed by the name assigned to the program or blank if no assignment. This provides a synopsis of all assignments and an aid to programl allocation during application development.

Languages : This page displays a list of up to 7 available languages.
Clicking on a language switches the user interface to the selected language.
Depending on the implementation, there are two possible language modes :

Static

- Only one language is concurrently supported.
- This language is selected at installation via a configuration option.
- This language selection page is not available.
- This mode is expected to be the general case, the only mode supported by some implementations.

dynamic

- The language can be switched during operation.
- There may be only one language at a time, valid for all users.
- Administrator privileges are required to switch the language.
- Text entered or edited by the user in the database is not changed (the database is unique and shared by all languages).
- Existing entries in the journal files are not changed (except for the state words, which are contained in a locale javascript table).
- This mode applies to advanced specific cases or demo configurations.

In fact, a user community is expected to use no more than two or three languages, or frequently only one, restricting this functionality to a mere demo object.

The language list is user definable and modifiable. It is contained in the record S_LANG of the Constants and Parameters domain (H) and is editable with the catch-all editor. To become operational, a new language must be added to this list and have a locale directory containing all translated files.

As some browsers keep javascript files in cache even when the calling HTML page is reloaded, switching to a new language may require the cache to be manually cleared (command available as CTRL/F5 on most browsers, but not availble to a web page for obvious reasons).

Home : This page displays a clickable list of up to 7 web pages. The selected page becomes the new home page. The list is contained in the record S_HOME of the Constants and Parameters domain (H) and is editable with the catch-all editor.

Version : This page displays an information message about the current server system.

3.9 Simulations

This entry of the main menu is only available to the system administrator. It provides access to the editing capabilities of domain I, the simulation values domain, and to possible other simulation scenarios (temporary and unsupported). Editing operations are identical as for programs (submenu **Editing**).

4 Processing units

4.1 Overview

Every channel may be configured to be the subject of one or several operations performed repeatedly by processing units (**PU**). PU are organized in a hierarchy, that is, if several PU apply to the same channel, they are executed in a strict **order** to make the latest results of earlier PU available to later PU.
Some PU are specific to numeric channels or to analogic channels, others are common to both, but are independent in the order of execution.
For each channel, the user decides the list of associated PU by crossing the appropriate check boxes on the PU page of the User Interface.
PU may conflict with each other or use a resource for different purposes : care must be taken not to apply these PU to the same channel simultaneously.

Each PU is rescheduled at some given time interval, according to its **sched** index :

Sched	Rescheduling interval
0	100 ms (clock tick)
1	1 second
2	9 seconds (100 times every 15 min)
3	1 minute
4	15 minutes (quarter hour)

4.1.1 Numeric channels

The value of a numeric channel is the value of its associated bit in the *I/O matrix*.
This bit can be either the image of a numeric HW input, which is volatile and likely to change state any time *(physical channels)*, or manipulated by software *(virtual channels).* Having both is not very meaningful and should be avoided (or restricted to testing/debugging).
When a PU is processed, the channel value found in the I/O matrix is used without any assumption on when and how it was set.
For numeric channels, *hardware value* and physical *value* are the same (or complemented if negative logic).

List of current processing units :

Order	Sched	Name	Function
0-1			Reserved
2	4	progdchan	Programmed channel
3	0	clonechan	Copy the value of a channel from its associated channel
4	0	dcombo	Logical combination of 2 channels
5	0	dtoggle	Channel toggling
6			Reserved
7	0	copychan	Copy the value of a channel to its associated channel
8	4	monitdchan	Monitored channel
9	0	dpulse	Pulsed numeric channel
10	0	dchange	Actions on numeric channel state change
11	4	dlogchan	Daily logging
12	0	copyseq	Copy the value of a channel to its seq channel
13-15			Reserved

4.1.2 Analogic channels

The *raw value* (or *hardware value*) of an analogic channel is the value in its associated word in the I/O matrix. It usually comes from a hardware sensor and in a few cases from a software source.

Its *physical value* is usually computed by applying a (most frequently linear) transformation to the raw value, but may also be copied from another channel or supplied by software. The *raw* and the *physical* values are integers, possibly rescaled to fit some storage size (byte or short integer). The physical values, although usually not directly shown to the user, are chosen to be consistent within a group of similar channels (i.e. handling quantities of the same kind and range) and to be directly comparable to channel independent threshold values. For example, the supported temperature range is as following :

Physical (unsigned integer)	1	45	85	127
Temperature (°C)	-22	0	20	41

Its *display value* may be its physical value or a linear transformation of the physical value in order to produce a presentation value meaningful to the user in terms of decimals and physical units. This transformation uses channel independent parameters (in the example : 0.5 and -22.5) and is performed by the client executing a Javascript procedure using physical values as input.

An analogic channel may also be a host, that is a repository of parameters and variables for a Processing Unit elaborating a more or less complicated behaviour involving other channels. In that case, it loses its analogic channel properties and some of the contents of its channel table are reassigned to different PU specific purposes.

List of current processing units :

Order	**Sched**	**Name**	**Function**
0			Reserved
1	0	avalue	Analogic channel value evaluation
2	4	progachan	Programmed channel
3	0	clonechan	Clone of associated channel *(same as numeric)*
4	0	dcombo	Logical combination of 2 channels *(same as numeric)*
5-6			Reserved
7	0	copychan	Copy the value of a channel to its associate channel
8-9			Reserved
10	0	achange	Actions on analogic channel state change
11	4	alogchan	Analogic channel daily logging
12	0	copyseq	Copy value to channel specified in Seq *(same as numeric)*
13	0	blink	Blinking a numeric output channel *(hosted)*
14-15			Reserved

4.2 Numeric channels

4.2.1 Channel table parameter names

The parameter names in the following tables are used in the description of the Processing Units. Some entries may have more than one name, they are synonyms identifying the same parameter used in different contexts .The description uses one of them, depending on the relevant context.

Name	**Contents**
Chan	Channel number
Ctype	Channel type
Cprio/Cnext	Priority
HWchan	Associated Hardware channel
Chan1/Prog	Associated channel 1 / Program
Chan2	Associated channel 2
MSGclus	Messages base
PUmask	Processing Units mask
Stabits	Static bits mask
Dynbits	Dynamic bits mask
Progval	Set point value
Chan3/Seq	Sequence / Channel 3 / Time unit
Chan4/Delta	Channel 4 / Increment

In the following, the owner of the PU is called the *master* if its value is used as input or the *target* if its value is set by the PU.

Names of the bits in the static bits mask

Name	Function
D_INITIAL	Value at startup
D_OFFLOG	Log ON→OFF transition
D_ONLOG	Log OFF→ON transition
D_HYST	Logical value of hysteresis zone
D_NOWRITE	Channel write protected
D_DAYLOG	Daily logging
D_USERCTL	User control
D_COMBO	Logical channel combination (0=OR, 1=AND)
D_LDSETVAL	Load static value from disk

4.2.2 progdchan - Programmed channel

A channel may be time programmed, with 96 different values in a day, one per quarter hour. The entry Prog of the channel table contains the number of a program within the program domain. This number is encoded as following :

bits	meaning
7-0	program number (2..252, if == 1, the feature is disabled. Note : 0 defaults to channel)
7	calendar usage flag : if (program >= 128) the value returned by the calendar function (not yet fully supported) is added to the program number (implemented : 0=working days, 1=Saturdays, 2=Sundays;considered : 3=user defined days) Good practice for programmed channels intending to use this feature : organize the upper half of the program domain in 4-program clusters aligned on multiples of 4.

4.2.3 clonechan - Copy the value of a channel from its associated channel

Chan1 → target
The value of the associated channel Chan1 is copied to the value of the target, which becomes a clone of Chan1. This allows to make more than one clone from the same associate.

4.2.4 dcombo - Logical combination of 3 channels

This unit combines 3 channels (Chan1, Chan2, Chan4) to generate the value of the target. The logical combination depends on flag D_COMBO (Logical channel combination in the static bits mask)

D_COMBO	HWchan	Logical combination
0	<128	Chan1 OR Chan2 OR Chan4
0	>=128	Chan1 XOR Chan2 XOR Chan4
1	x	Chan1 AND Chan2 AND Chan4

Action depends on the type of the target :
- numeric : the value becomes the new value of the target
- analogic : if value=0, the target value is cleared, else, it remains unaffected

If an input channel is not used, its number must be set to 0 for OR (always returns 0) and 128 for AND (always returns 1). For XOR, it may be either, depending on the wanted result.

4.2.5 dtoggle - Channel toggling

This unit has two flavors, depending on having one or two input channels (Chan1, Chan2) configured :

- one channel (Chan2 set to 0) :
 a 0 → 1 transition on Chan1 toggles the target.

- two channels :
 a 0 → 1 transition on Chan1 sets the target (precedence)
 a 0 → 1 transition on Chan2 resets the target
In any case, 1 → 0 transitions are ignored.

4.2.6 copychan - Copy the value of a channel to its associated channel

master → Chan2
The physical value of the master becomes the physical value of Chan2.
If Chan2 is numeric and the master is analogic and in the hysteresis zone, the last master's state outside the hysteresis zone applies.
If Chan2 is numeric and has bit 7 set, the master's value is complemented before use.
This works also if Chan2 has no channel table.

It may be used to duplicate an analogic channel if both the master and Chan2 are analogic.
Copying numeric to analogic is a special case : Chan2 is left unchanged if the master is set and cleared otherwise.

4.2.7 monitdchan - Monitored channel

A channel may be time monitored, with 96 different values in a day, one per quarter hour. This is similar to progdchan, except that instead of setting, the value of the channel is compared to the programmed value. If they are equal, Chan4 (the « alarm output ») is reset, else Chan4 is set and the « Action on change » function is called to evaluate the system's reaction.
Chan2 is used as an enable-disable input.

4.2.8 dpulse - Pulsed numeric channel

On normal operation, this channel delivers a pulse of specified constant width when the logical combination of the 2 control channels (Chan1 and Chan2) goes high, regardless whatever this combination does afterwards (going low during the pulse or staying high after the pulse).
To deliver a new pulse, both input and output must have gone low.
This PU is incompatible with those using the same resources.

Resources:

HWChan	pulsed output		
Chan1	control input channel 1	logical combination	
Chan2	control input channel 2	selected by D_COMBO	
Progval	pulse length (in time units specified in Delta)		
Delta	time unit (**sched** index)		
D_COMBO	Chan1 Chan2 logical combination selector (0=OR, 1=AND)		
D_HYST	output channel value when idle		
	0 true pulse	input: ___----	output: ___--__
	1 delayed command	input: ___----	output: -----__

4.2.9 dchange - Actions on numeric channel state change

These actions depend on the values of flags D_ONLOG, D_OFFLOG, D_ONSEQ, D_OFFSEQ

If D_ONLOG or D_OFFLOG set, the corresponding state change (OFF□ON or ON□OFF)
triggers the insertion of a status message into the log file(s), with, as parameters,
the channel number and its final state.

If D_ONSEQ or D_OFFSEQ set, the corresponding state change starts the associated sequence.

Associated state words :

0 ON → OFF	0 OFF	2 OPEN	4 OUT OF ORDER	10 END
1 OFF→ ON	1 ON	3 CLOSED	5 WORKING	11 START

4.2.10 dlogchan - Numeric channel daily logging

Every 9th second, in the logging buffer, the byte indexed by the current quarter hour is incremented
if the channel is set. This results in a value in the range 1..100 representing the % of 15 mn time during which the channel was set, which increases the precision of the graphical display.

4.2.11 copyseq - Copy the value of a channel to its seq channel

master → Chan3
This is the same as copychan, except that the Chan3 (Seq) number is used as associate channel number. This would be incompatible with sequence management in numchange and pulsed channel

4.3 Analogic channels

4.3.1 Channel table parameter names

The parameter names in the following tables are used in the description of the Processing Units. Some entries may have more than one name, they are synonyms identifying the same parameter used in different contexts .The description uses one of them, depending on the relevant context.

Name	**Contents**
Chan	Channel number
Ctype	Channel type
Cprio/Cnext	Priority
HWchan	Associated Hardware channel
Chan1/Prog	Associated channel 1 / Program
Chan2	Associated channel 2
MSGclus	Messages base
PUmask	Processing Units mask
Stabits	Static bits mask
Dynbits	Dynamic bits mask
Progval	Set point value
Chan3/Seq	Sequence / Channel 3 / Time unit
Chan4/Delta	Channel 4 / Increment
Coefa	Conversion coefficient A
Coefb	Conversion coefficient B
Coefc	Conversion coefficient C
Hyst	Hysteresis

In the following, the owner of the PU is called the *master* if its value is used as input or the *target* if its value is set by the PU.

Names of the bits in the static bits mask

Name	**Function**
A_NLOWLOG	Log normal→LOW crossing
A_LOWNLOG	Log LOW→normal crossing
A_NHIGLOG	Log normal→HIGH crossing
A_HIGNLOG	Log HIGH→normal crossing
A_NOWRITE	Channel write protected
A_DAYLOG	Daily logging
A_USERCTL	User control
A_OUTCHAN	Action on threshold crossing
A_OUTMODE	Action mode on threshold crossing
A_COMBO	Logical channel combination (0=OR, 1=AND)
A_LDSETVAL	Load setpoint value from disk

4.3.2 achange - Actions on analogic channel state change

An analogic channel is considered a three state device depending on its value being below, within (=normal) or above the normal range (also called hysteresis zone), which is comprised between a LOW threshold (setpoint - hysteresis) and a HIGH threshold (setpoint).

Each of the 4 possible transitions LOW ←→ Normal ←→ HIGH may conditionally cause any or all of the following:

- a log message to be issued
- a sequence to be started
- one or two numeric channels to be set or cleared

The first of a pair of numeric channels is Chan2. The second is Chan2+1
Action mode is selected by flag A_OUTMODE :
A_OUTMODE 0=one channel, 1=two channels, according to the following truth table :

A_OUTMODE	**0**	**1**	
	Chan2	Chan2	Chan2+1
HIGH	OFF	OFF	ON
Hyst zone	no action	OFF	OFF
LOW	ON	ON	OFF

The action outside the hyst zone is inverted if Chan2 has bit7 set.

Flag A_OUTHYST retains the last transition normal → LOW (=0) or normal → HIGH (=1) when the channel is back to the normal (hysteresis) zone. This is assumed to be the 'numeric value' of the analogic channel in situations where a binary ON/OFF value is required.

4.3.3 *alogchan - Analogic channel daily logging*

A new value is written to the log buffer every quarter hour. It is calculated by adding up the current value every 9 sec (100 times during the quarter hour) and dividing by 100 at the end of the quarter hour.

4.3.4 *avalue - Analogic channel value evaluation*

Linear conversion : the hardware value is converted into a physical value with the formula :

$$y = ((a*x)/c)-tofbase+b \text{ with } y >= 0$$

If c=0, c=1 is used.

As this is integer arithmetic, a/c allows to define fractional multipliers in the range :

{ 0, 1/255=0.003921, .. 255/1 }

4.3.5 *blink - Blinking a numeric output channel*

This is an example of an analogic channel hosting a non-analogic function.
It is triggered by the AND or OR logical combination of 2 numeric inputs, produces a given number of pulses on a numeric output (with user programmable parameters pulse length and interval), followed by a longer pulse on the same output, then by a "final action" pulse on a second numeric output.

Example of time diagram : assume 4 pulses and a last long pulse :

```
blink sequence______--______--______--______--______-------------___________
final action  ________________________________________________--------______
```

An alternate configuration makes the blinking last forever.

Resources :

HWchan	final action output channel
Chan1	control input channel 1
Chan2	control input channel 2
Chan3	pulsed output channel
Delta	timelength multiplier for the last warning pulse
Ctype	number of pulses before final action
Coefa	warning pulse length (in ticks)
Coefb	interval between warning pulses (in seconds)
Coefc	timelength of the final action pulse (in ticks)
D_COMBO	Chan1 Chan2 logical combination selector (0=OR, 1=AND)

4.3.6 *progachan - Programmed channel*

A channel may be time programmed, with 96 different values in a day, one per quarter hour. After rescaling, the current value is written to the Setval entry of the channel table.
The entry Prog of the channel table contains the number of a program within the program domain of the data base. This number is encoded as following:

bits	meaning
7-0	program number (1..255, 0 defaults to the channel number)
7	calendar usage flag : if set, the value returned by the calendar is added to the program number (supported : 0=working days, 1=Saturdays, 2=Sundays ;considered : 3=user defined days) A set of 4 consecutive program entries must be allocated for each channel using this feature (the same set may be shared by several channels with identical programming requirements) Note: the programs in the range 1..127 are not affected and should be used if this feature is not required.

5 Sequencing

5.1 Overview

Some applications require operations where a number of outputs are switched in a precisely timed succession or where the system's behaviour depends on the logical combination of several input channels. The sequencing support was designed to answer requirements exceeding the functionalities described so far.

This support is implemented by a combination of a *sequence engine*, *programs* and *sequences*.
The sequence *engine* is the piece of software that interprets and executes a program.
A **program** is a read-only list of instructions written in the *Sequence Language*. This list is stored in one record of the Sequential Programs domain in the data base, meaning that it cannot exceed a length of 128 bytes. The number of programs is currently limited to 60.
A **sequence** is a unit of execution of a program. It includes a sequence parameter table indicating which program is to be executed, an execution status flags mask and a *context descriptor* consisting in several constants and variables.
Several sequences may execute the same program using different context descriptors.
The *Sequence Language* is composed of verbs of one alphabetic letter, each representing an elementary operation such as testing an input, switching an output, waiting some time or branching.
An instruction of the language is made of one letter optionally prefixed by a decimal number and one or two parameter symbols (non-alphabetic characters), whose sum is used as *argument* when the instruction is executed (this sum is termed the *full evaluation* of the argument).
The user has access to creating, editing and deleting programs and sequences.

5.2 Sequences

A program is executed in the context of a sequence. This context is contained in a *Sequence Control Block* (**SCB**) , a structure in memory, which includes two sets of parameters :

- the *dynamic paramers*, which vary during the execution of the sequence. They are not directly user accessible
- the *static parameters*, whose values can be edited by the user and do not vary during the execution of the sequence (with a few exceptions noted later).

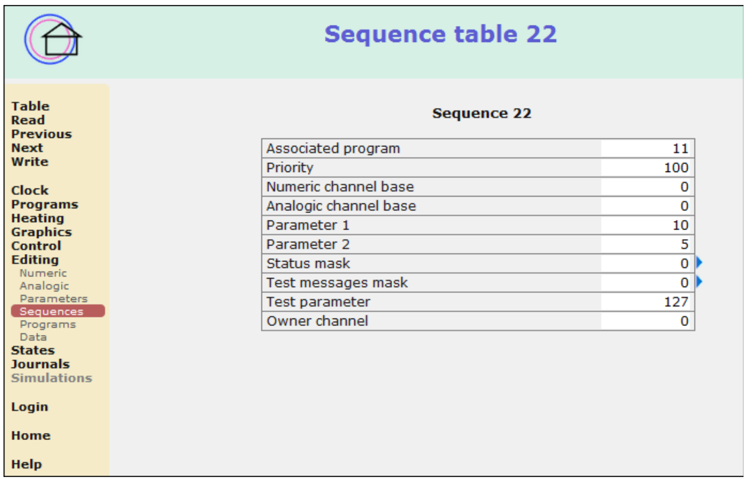
SCB structure (static parameters)

Name	**Description**
Seq	sequence number. Number of the record in domain M.
Prog	associated program. Number of the executed program. If 0, the sequence is not loadable.
Prio	priority within the other loaded sequences. This defined the order of execution of the sequences. If 0, the sequence is not loadable.
Base0	numeric channel base. Conditionally added to any argument representing the number of a numeric channel.
Base1	analogic channel base. Conditionally added to any argument representing the number of an analogic channel.
Param1	general purpose parameter 1
Param2	general purpose parameter 2
Stabits	status mask. Records the execution status of a sequence.
Debug	test messages mask. Debugging aid, disabled in normal operation.
Testpar	test parameter. Debugging aid, unused in normal operation.
Owner	owner channel. Number of the channel that started the sequence. User defined if the sequence is not started by a channel.
Usarg	value passed when the sequence is started. This depends on the activation mechanism and the event that caused the start. This is not editable.

The SCB is allocated in memory at startup and initialized with data from its image on disk, contained in one of the records of domain **M** (this is called *loading* the sequence). The SCB is user editable any time with the Sequence Editor, except if the sequence is active. Changes on disk are reflected in memory at the end of an editing session. If the sequence has become loadable, it is created. An existing sequence can be deactivated but not removed from memory unless the application is restarted.

Any number of sequences may run concurrently provided they do not perform conflicting operations or create deadlocks (no general protection mechanism is implemented)

Up to 60 sequences may be created.



Sequence 22

Associated program	11
Priority	100
Numeric channel base	0
Analogic channel base	0
Parameter 1	10
Parameter 2	5
Status mask	0
Test messages mask	0
Test parameter	127
Owner channel	0

SCB editor

Sequence start methods :

Value of Usarg	**Start method**
1	user command
Usarg of parent	another sequence (the *parent*)
1	application startup (sequence set active on disk)
1	numeric channel transition ON -> OFF
2	numeric channel transition OFF ->ON
1	analogic channel transition NORMAL -> LOW
2	analogic channel transition LOW -> NORMAL
3	analogic channel transition HIGH -> NORMAL
4	analogic channel transition NORMAL-> HIGH

Sequence termination methods :

Value of Usarg	**Termination method**
0	exiting
0	user command
0	another sequence

5.3 Programs

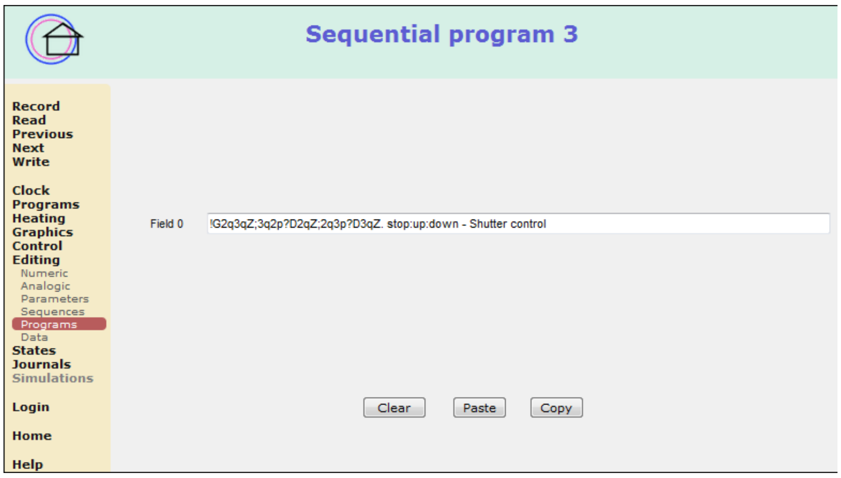
A program is a list of executable statements (also called instructions) written on one line and stored on disk as a record in domain **N**, with a copy in memory if it is used by any loaded sequence. As for sequences, it is user editable and is loaded in memory at application startup or when the first sequence using it becomes loadable.
It is executed from left to right, except if a backward branch occurs.
A statement is composed of an instruction verb (an alphabetic character) prefixed with a string of non-alphabetic characters which eventually evaluates to a numeric value, the argument of the instruction verb, which is used by the sequence interpreter to perform the action represented by the verb.
Example :

27T	Test channel 27, if set then skip the next instruction, else execute it

As in this example, for many of the verbs, the argument represents a channel number. These verbs may be written either in upper or in lower case, with the following difference :

27T	perform the test on numeric channel 27 (take 27 as an absolute value)
27t	add Base0 to 27 before performing the test
78T	perform the test on analogic channel 78 (take 78 as an absolute value)
78t	add Base1 to 78 before performing the test

The channel bases must be chosen such that the final channel number still refers to the correct channel type, numeric or analogic.
For verbs using an argument which is not a channel number, upper and lower case are different verbs.



Field 0 `!G2q3qZ;3q2p?D2qZ;2q3p?D3qZ. stop:up:down - Shutter control`

Program editor

5.4 The Sequence Language

The sequence language features various types of actions:

- reading and testing a channel
- writing a value to a channel
- delays and timers
- flow control (branch, loop and subroutine call)
- setting, testing and logically combining internal variables
- acting on other sequences
- conditionally issuing messages (monitoring and debugging)

5.4.1 Operands

A program is a list of statements, which have the following format:

`<decimal value><operand><operand><verb>`

All components but the verb are optional. Any missing component defaults to 0.
The decimal value is any string composed of decimal digits making a number, which must be comprised in context dependent limits (Example: 1..127 if a channel number).
An operand represents the value stored in an associated SCB member (which may be hidden) :

Operand	Represented SCB member
!	Usarg (value passed at sequence start)
?	Param1 (general purpose parameter 1)
$	Param2 (general purpose parameter 2)
#	LEFT register
+	increasing index
−	decreasing index

The argument value used to evaluate the action of the verb is the arithmetic sum of the decimal value and the operands. Example:
Assume SCB members *Param1* and *Base0* contain the values 5 and 3 respectively, the statement

`21?p`

would mean : set channel 29, the value of the argument being 21 + 5 + 3
(note that **`21?P`** would set channel 26, as no base would be applied).
If the argument evaluates to 0, this may have a special meaning for the action of the verb.

5.4.2 Registers LEFT and RIGHT

The language implements two internal registers called LEFT and RIGHT, designed to allow operations using the values of analogic channels. Numeric channels would be no special case, their analogic value would be 0 or 1. These registers can be loaded from or written to a channel and compared to each other with the following logical operators :

Operator	Logical operation	
|	OR	
&	AND	
<	LEFT less than RIGHT	)
=	LEFT equal to RIGHT	) arithmetic comparison

>	LEFT greater than RIGHT)
^	XOR
3~ or 4~	logical value of LEFT or RIGHT

The result of the operation, reduced to a logical value (0 or 1), is stored in the internal volatile variable STATUS used by the immediately following statement, usually a test for true (`T`) or false (`F`) without argument. Example :

`&F`	test if the operation LEFT AND RIGHT yields 0

Note that there is no operator for 'less or equal' or 'greater or equal' as they can be implemented by their opposite operator ('greater' and 'less') while exchanging `T` and `F`.

Additional operations on LEFT and RIGHT are provided by a special verb (~)

Statement	**Operation**
0~	swap the contents of LEFT and RIGHT
1~	LEFT = RIGHT (overwrite LEFT with the contents of RIGHT)
2~	RIGHT = LEFT (overwrite RIGHT with the contents of LEFT)
3~	logical value of LEFT (same as '&' with RIGHT=1)
4~	logical value of RIGHT (same as '&' with LEFT=1)

5.4.3 List of verbs

a) Upper case (and dependent lower case)

Verbs whose argument is a channel number (and have a dependent lower case) are flagged with an asterisk. The value of the fully evaluated argument is designated by **n**.

		(space) NOP between instructions, Error if within an instruction.
@		subroutine entry point. Illegal within a program
A		debugging message
B		backward branch to the n-th target (:) relative to B's position. If 0, no branch.
C	*	write the complement of STATUS to channel
D		inline delay. Wait here until delay elapsed. If 0 no wait.
E		skip if sequence n is idle or not existing or myself.
F	*	skip if false (see also T)
G		forward branch to the n-th target (;) relative to G's position. If 0, no branch.
H		load Param1 with n (h=load Param2). Initial values are restored on new start.
I		
J	*	write an entry to the journal file.
K		start sequence n (k=kill sequence n)
L	*	load LEFT with channel logical value (0 or 1, works with all channels)
M	*	write LEFT to channel n (works with all channels)
N	*	add value of channel n to RIGHT, clear RIGHT if n=0
O		
P	*	write 1 to channel (ON)
Q	*	write 0 to channel (OFF)
R	*	load RIGHT with logical channel value (0 or 1, works with all channels)
S	*	write RIGHT to channel n (works with all channels)
T	*	skip if true (see also F)
U		
V		load LEFT with n (v=add n to LEFT)
W		init timer (w=test timer)
X		load index with n (x=decrement index and skip if 0)
Y		
Z		sequence exit (z=return from subroutine)
.		(dot) program end. Anything following is ignored and may be used as comment.

b) Independent lower case

g	call the subroutine starting with the n-th subroutine entry (@). If 0, no call.
h	load Param2 with n (H=load Param1). Initial values are restored on new start.
j	trace message for test
k	kill sequence n (K=start sequence n)
v	add n to LEFT (V=load LEFT with n)
w	test timer and skip if elapsed (W=init timer)
x	decrement index and skip if 0 (X=load index)
z	return from subroutine (Z=exit sequence)

Notes:

- Letters with a blank description or not present in the above tables are invalid verbs and generate an error and a forced exit.
- The state of channel 0 is always 0
- Writing to channel 0 has no effect

5.4.4 Flow control (skipping and branching)

There are two branch operations : backward (B) and forward (G). The targets are respectively the colon (backward) and the semi-colon (forward). The branch argument is the number of targets to be counted self relatively to the verb to find the actual target. Example:

Program	Detail	Description
`1G:5L:;6R2B`	`1G`	forward branch to the first semicolon after G
	`:`	target for a B instruction (here the target of the last 2B instruction)
	`5L`	some operation
	`:`	target for a B instruction (unused in this example)
	`;`	target for a G instruction (here the target of the first 1G instruction)
	`6R`	some operation
	`2B`	backward branch to the second colon counting backward from B

A backward branch instruction is used to implement a loop and is usually (but not necessarily) immediately preceded by some kind of test operation, which causes the next instruction to be skipped (success) or executed (failure).
A forward branch instruction is used to conditionally skip or execute some part of the program located on its right.
The targets are ignored by execution of the program and can be used as NOPs, as well as the instructions 0B and 0G.
In addition to the test instructions T and F, the timer w, the index instruction x and the sequence state testing instruction E may cause a skip.
Note : Take care if targets are added or removed on program modification, as this could impact the arguments of B and G instructions.

5.4.5 Delays and timers

The **delay instruction (D)** causes the sequence to wait for the time specified as argument. There is no means to shorten this delay, except killing the sequence. The argument is a 3-digit decimal number where the leftmost digit specifies a time unit for the number 1-99 of the two other digits:

Leftmost digit	**time unit**	**delay range** (rounded)
0	100 ms	9 s
1	1 s	1mn 30 s
2	9 s	14 mn
3	1 mn	1h 30 mn
4	15 mn	24h

Notes :

- the smallest time granularity is 100ms
- leading zeros can be omitted (7 or 07 or 007 all represent 700 ms)
- if the time is 0, there is no delay. This is the case for the argument values 0, 100, 200…
- if a long delay requires a better precision than allowed by its time unit, this can be achieved by adding two or more delays with different time units.

The **timer instruction (W and w)** allows to implement a time-out loop : use W to set the time in front of the loop and w to control the loop. For example, assume you want to wait 30 minutes for channel 21 to become ON :

```
330W:21F1Gw1B <code if timeout first> ; <code if channel ON first>
```

Program	Detail	Description
`330W:21F1Gw1B...;...`	`330W`	set the timer to 30 mn
	`:`	target for the 1B instruction
	`21F`	test channel 21 and skip if OFF
	`1G`	leave the loop if channel 21 becomes ON
	`w`	test the timer. If elapsed, skip next instruction
	`1B`	branch to the first target on the left

5.4.6 Indexing

The SCB includes a member used as index. It is loaded with the argument value n of an **instruction X** and decremented by **instruction x**, giving a skip when 0. This is used for implementing a loop executed a given number of times. The index may appear in an argument as one of the characters + or −, termed *increasing index* and *decreasing index*. The first (+) varies from 0 to n-1, the second (−) varies from n-1 to 0. Example :

`6X:20+Px1BZ`	set channels 20 to 25 to ON

5.4.7 Sequence synchronisation

Sequences can run simultaneously and act on each other : a sequence can start another sequence (**instruction K**) or kill a sequence (**instruction k**). A sequence started by another sequence inherits the *Usarg* value of its parent.

When executing concurrently, two sequences may perform conflicting operations, like one opening a door, the other closing it. This can be avoided by having each sequence testing that the other is inactive (**instruction E**) or that a conflicting command is not in progress (**instructions T or F**).

5.4.8 Exit and kill

The **exit instruction (Z)** immediately terminates the current sequence. It is the explicit responsibility of sequence to leave a clean state when exiting.
The **kill instruction (k)** terminates another sequence. This sequence may leave a 'dirty' state, like letting ON a channel that should be OFF. To solve this problem, the killed sequence is restarted with parameter *Usarg* set to 0 (A normal start never passes this value). The sequence should then execute some cleanup code and exit.
All sequences that are active long enough to be possibly killed should start as following :

```
!G <cleanup code>Z ; <normal code>
```

This ensures that the cleanup code is executed on kill. This code may be empty, but the **Z** instruction must be present. Killing an inactive sequence has no effect.

The **dot instruction (.)** should be the last statement in any program. Control should never reach it, as this would be a programming error, for example a missing **B** or **Z** instruction or branch target. If executed, it generates an error and a forced exit. Anything following the dot is ignored and treated as comment.

5.4.9 Subroutines

A subroutine starts with a @ and returns with a **z**. It is called by the main program with a **g**, after which execution resumes when the subroutine returns.
Subroutines must be located one after the other after the end of the main program. They cannot be nested or call each other, as only one call depth level is currently supported. A @ appearing within the main program or a subroutine generates an error and a forced exit. The first @ ends the main program. If there are several subroutines, each subroutine ends with the '@' starting the next. The final dot must end the last subroutine. Example :

`!G1gZ;<normal code>Z@21Qz.`	the cleanup code calls the subroutine turning channel 21 OFF

5.5 A basic example : a simple rolling shutter control

Basic specification :

- o the application uses channels 21 to open and 31 to close the shutter
- o the shutter features 'full open' and 'full closed' end switches that automatically limit its movement
- o a full open or close operation takes a maximum of 20 s.
- o this should be the maximum closing time of the power switches to prevent their overheating
- o an open or close operation can be started, interrupted and restarted any time by the user

Implementation : a single program executed by 2 sequences, one for opening, the other for closing.

The program (assume its number is 3) :

```
!G21qZ;21p120D21qZ.
```

Detail	Description
`!G`	for start/stop discrimination
`21qZ`	cleanup code on stop (kill) command : channel 21 set to OFF and exit
`;`	target of the !G instruction on start command
`21p`	channel 21 set to ON
`120D`	20 s delay
`21q`	channel 21 set to OFF
`Z`	exit

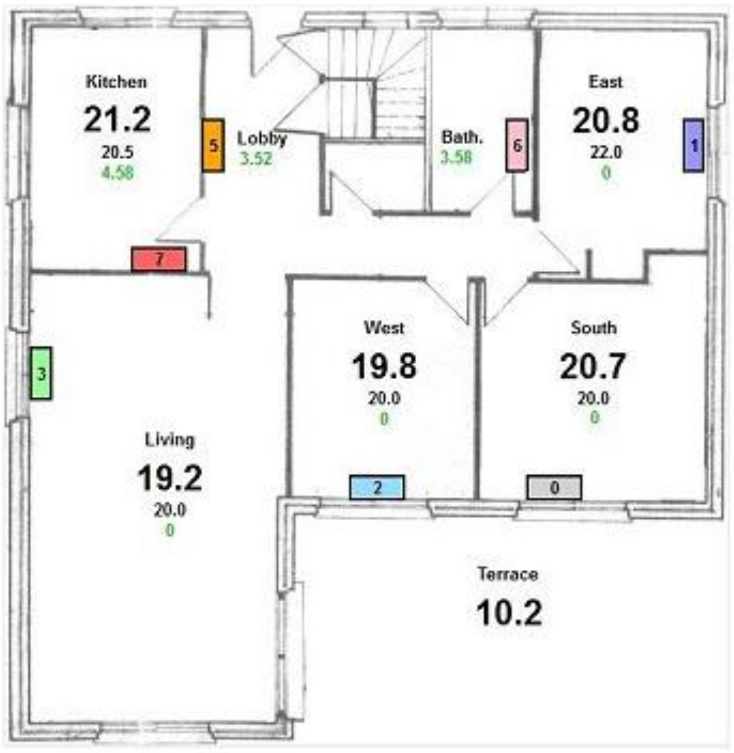
The SCP parameters of the 2 sequences :

Sequence	**Open**	**Close**
Seq (user chosen)	10	11
Prog	3	3
Base0	0	10

The channel switching instructions used in the program (**p** and **q**) are lower case, meaning that Base0 is added to their arguments : channel 21 in the Open sequence becomes channel 31 in the Close sequence.

6 Application examples

6.1 Heating regulation

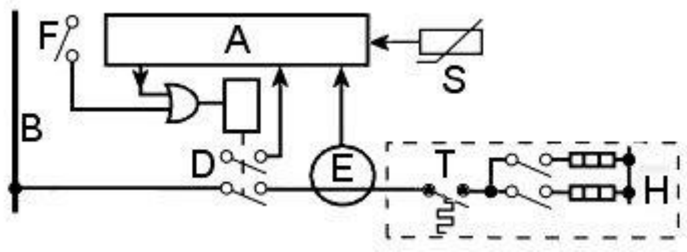


Assume a house with a living room, three bedrooms and a kitchen, all equipped with electric heaters. You want to have an independent temperature regulation system for each room.
The user interface would be based on a schematic layout of the house, showing a set of relevant information for each room :

Name of the room
Current temperature
Required temperature
Electric consumption

The lobby and the bathroom have a heater, but no independent regulation. There is an outside temperature sensor on the terrace.

Each heater tells its current state by a number (0 to 7) and a corresponding color.
Each room is provided with the following heating circuitry :



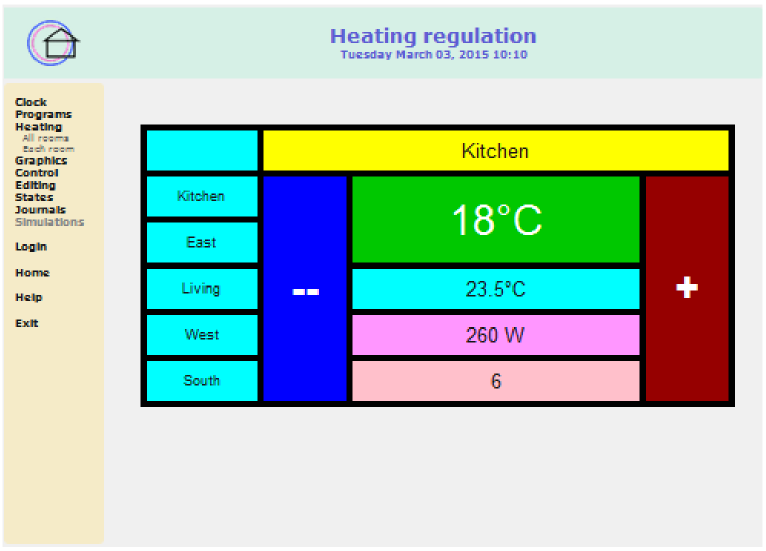
The heater H is connected to the mains B through a power switch D and a current sensor E.
The power switch D is operated either by the computer A or manually via the switch F (logical OR).
The heater has a local thermostat T and one or several heating elements that may be locally switched ON or OFF.
The power switch D has an auxiliary contact whose state is reported to the computer.
The current sensor E supplies an analogic value, proportional to the current.
An independent sensor S reports the room temperature to the computer .
The state of a heater is represented by a 3-bit value :

bit 0 : command issued by the computer to the power switch
bit 1 : state of the power switch (0=open, 1=closed)
bit 2 : current flowing through the heater (0=no, 1=yes)

Some of these 8 states are normal operation, others show a malfunction :

State	2	1	0	Meaning
0	no	open	OFF	all OFF
1	no	open	ON	power switch not working
2	no	closed	OFF	power switch: F ON;heater: local thermostat or switches OFF
3	no	closed	ON	heater: local thermostat or switches OFF
4	yes	open	OFF	incorrect power switch state reporting (F ON)
5	yes	open	ON	incorrect power switch state reporting
6	yes	closed	OFF	switch: F ON
7	yes	closed	ON	all ON

This kind of page may look too complicated for the user as well as for the developer.The same information may be provided in a simpler format :



In this implementation, each room has ist own view, identified by the top caption and selected in the left column. The temperature set point, in the light blue central rectangle, can be modified by clicking within the large plus/minus boxes.

6.2 Garage door automation

Assume an electric garage door that is operated (before being automated) by a push button, which could be a logical OR of a physical button (inside the house), a numeric keypad (on the outside), and a remote control. Once automated, the open door would close after a user defined delay with preliminary repeated acoustic warnings (beeps) to make people aware that they have to stay clear from the door or to give them the possibility to cancel the operation.

The automation would add :

- a user operated switch enabling or disabling the function (input channel)
- a door closed sensor (input channel)
- an additional switch closing the door (output channel)
- a warning beeper (ouput channel)

This functionality would be implemented by a « beeper » Processing Unit providing the following timed action :

```
beeper                 __--_______--_______--_______--_______--_______-------------_________
door closing command   ____________________________________________________________--------___
```

where all channels and timings are user definable parameters.

6.3 Lighting control

Corridor lighting

- switched on by a user push button (input channel)
- switched off after a user defined delay (pulsed output channel)

Outside lighting

- switched by an output channel either programmed or controlled by an input channel triggered by a dusk photocell

6.4 Intrusion detection

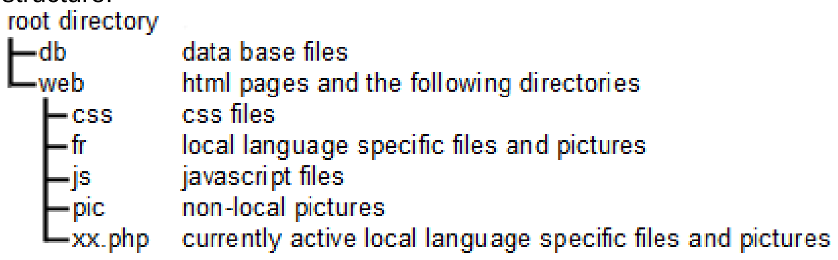
- detected by some sensing device (electromagnetic, volumetric, infrared) triggering a monitored input channel
- action by one or several output channels (lighting, sound alarm, door locking..) controlled by the input channel
- event journalling

7 Software installation

7.1 Organization on disk

7.1.1 Data files directory tree

Once the software is completely installed, all data files reside in a directory tree with the following structure:



```
root directory
├db          data base files
└web         html pages and the following directories
  ├css       css files
  ├fr        local language specific files and pictures
  ├js        javascript files
  ├pic       non-local pictures
  └xx.php    currently active local language specific files and pictures
```

In Raspbian, the root directory is named `www` by convention and is a first level subdirectory of `/var`.
In Windows, its path and name can be freely chosen, but should be `www` for consistency.
`db` contains all data base files, with names in upper case and the file type `.DB` .
Each of these files contains one or several domains:

File	Domains	Usage
`ZX.DB`	@ABCDEFIJKLMN	Tables, programs, macros, sequences
`UAD.DB`	G	User profiles
`MSG.DB`	H O P U	Text dictionaries
`L0.DB`	Q	Journal 0
`L1.DB`	R	Journal 1
`L2.DB`	S	Journal 2
`L3.DB`	T	Journal 3

`web` contains the `.htm` files and several subdirectories making up the user interface. It is the root of the integrated web server. For modifying the functionality of this interface or extending it to new user applications, files and directories may be modified or added manually to this space.
There can be one or several local language directories such as `fr`, each including all the files containing local language elements like text or symbols. They typically contain word dictionaries, translated help files, a `pic` subdirectory with all localized pictures and a text file called `L`, with a single text line containing the 2-letter language name, the same as the name of the directory. All these files must be translated if a new local language is added.
The directory of the currently active local language (english by default) is renamed to `xx.php` (but is renamed to the contents of its file `L` when another language is made active). On the demo version written in the PHP language, it is not a directory, but a PHP script which redirects the file fetch to the active language directory (which keeps its 2-letter name).
For a maximum of portability, all file and directory names are in the 8.3 format, except for files which are specific to a given architecture or system, where the local conventions apply.

7.1.2 Configuration and executable files

7.1.2.1 Raspbian

On Raspbian they are installed in the standard system directories for a daemon:

`/etc/zdanetix/zdanetix.conf`	configuration file
`/etc/init.d/zdanetix`	service startup/stop script
`/usr/sbin/zdanetix`	executable file

7.1.2.2 Windows

On Windows, the service script does not exist, the configuration and the executable files must reside in the same directory, which can be freely chosen.

7.2 Installation

7.2.1 Raspbian

7.2.2 Windows

There is currently no specific installation procedure.

8 Configuration

8.1 GPIO control

8.1.1 The GPIO header of the Raspberry Pi

Let's have another look at the correspondence between the BCM pins and the Raspberry Pi GPIO header pins :

7	6	5	4	3	2	1	0	Bank
26	31	29	7	5	3			8
10	8	33	32	23	19	21	24	9
16	15	40	38	35	12	11	36	10
				13	37	22	18	11

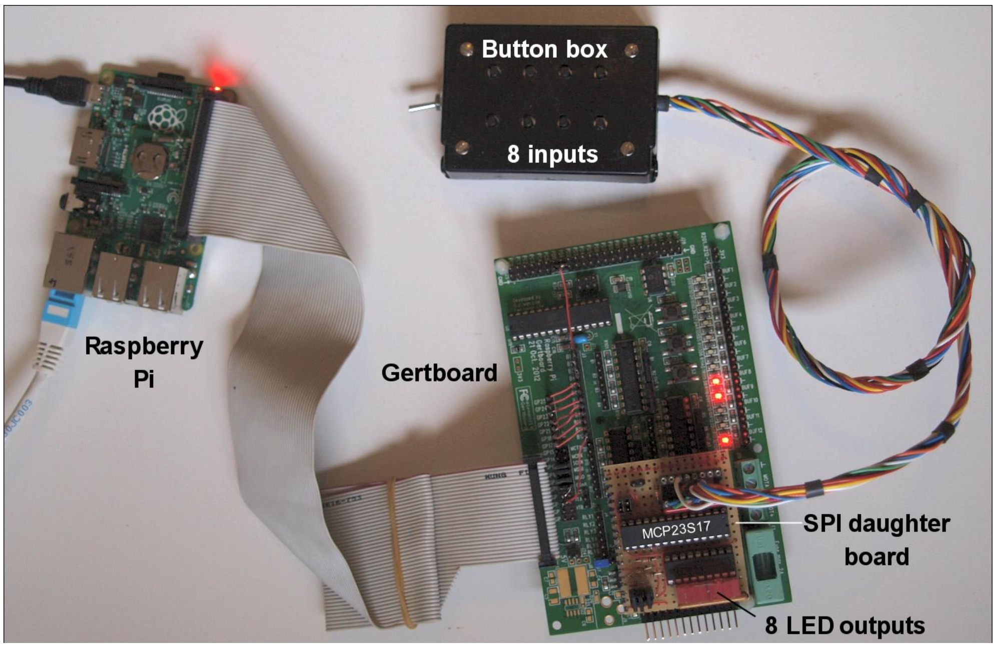
This table lays out the BCM pins as 4 banks of 8 pins, with the cells filled with the numbers of the RPi pins. The green background denotes the reserved pins (I2C, Serial, SPI), which we leave alone. The orange background denotes the 8 pins available on the 'old' RPi A or B (limited to 26), which we are going to use. Of course, the 9 remaining pins (white background) are subject to the same configuration rules as the first 8 pins.
Remember that when talking about pin numbers, we always refer to the BCM pins. If you have RPi pins in mind, you must always translate them to BCM pins yourself.

8.1.2 The Gertboard

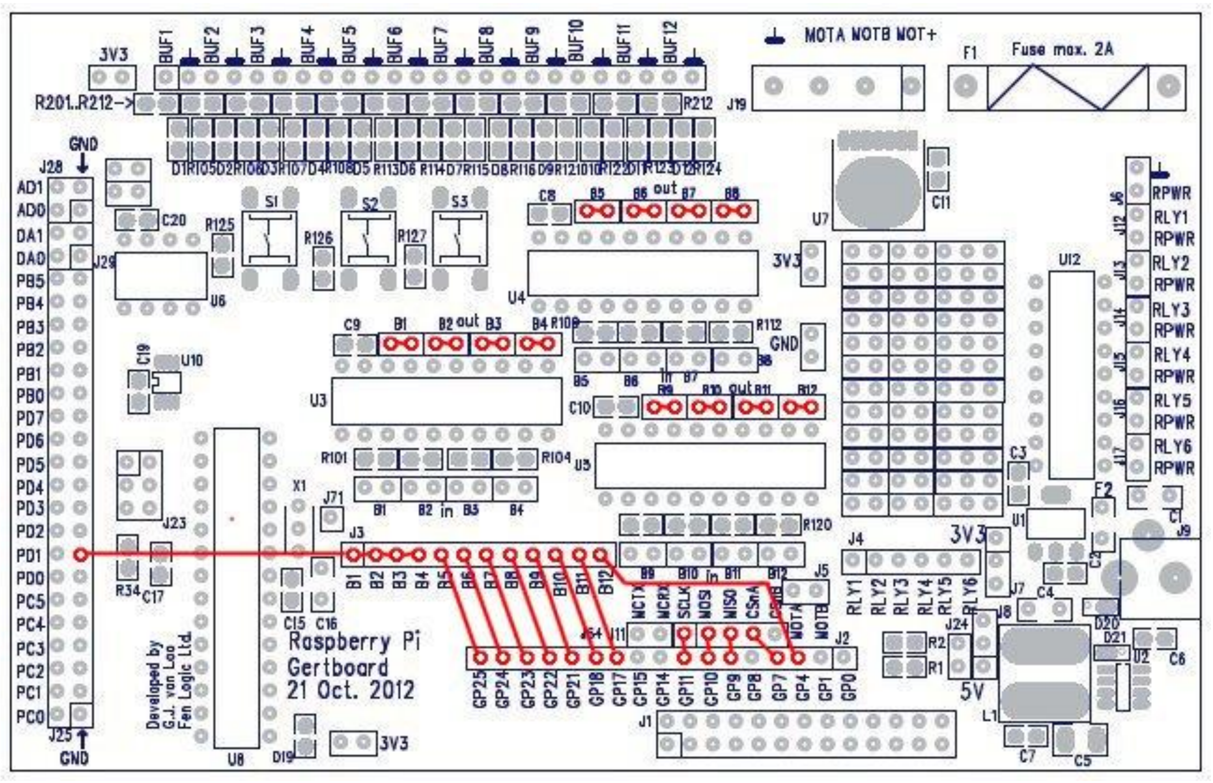
To demonstrate the operation of the RPI GPIO pins, we are using the Gertboard, which has become something as a standard in RPi I/O interfacing and is easily available.
Please refer to its user manual.

The used board was somewhat modified to

- accept a connection to the RPi via a ribbon cable,
- support a small daughter board featuring an MCP23S17 SPI I/O expander used concurrently with its MCP3002 A/D converter. It uses Chip Select CS0; CS1, which is normally assigned to the onboard D/A converter (which is not used here), is redirected to the A/D converter. More details later, when configuring SPI numeric I/O.



The wiring of the Gertboard is as following:



The outputs B1-B4 are disabled, the others are connected as following:

	B5	B6	B7	B8	B9	B10	B11	B12
BCM	25	24	23	22	27	18	17	4
RPi	22	18	16	15	13	12	11	7

8.1.3 GPIO outputs configuration

The GPIO configuration is done by editing nodes in domain H (Constants and parameters) with the Catch-All editor. A number of nodes are preallocated in domain H, starting with the node whose number is specified in Field 6 of record 254 in domain H (factory setting: 217)

For the time being, assume we want to configure all 8 pins as outputs (all jumpers on the output side on the Gertboard). This is the contents of the nodes we would have to program :

Node	Field 0	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
1	217	0	8	0	0x10	0	0
2	218	1	10	0	0xC6	0	0
3	219	2	11	0	0x0B	0	0
4	0	0	0	0	0	0	0

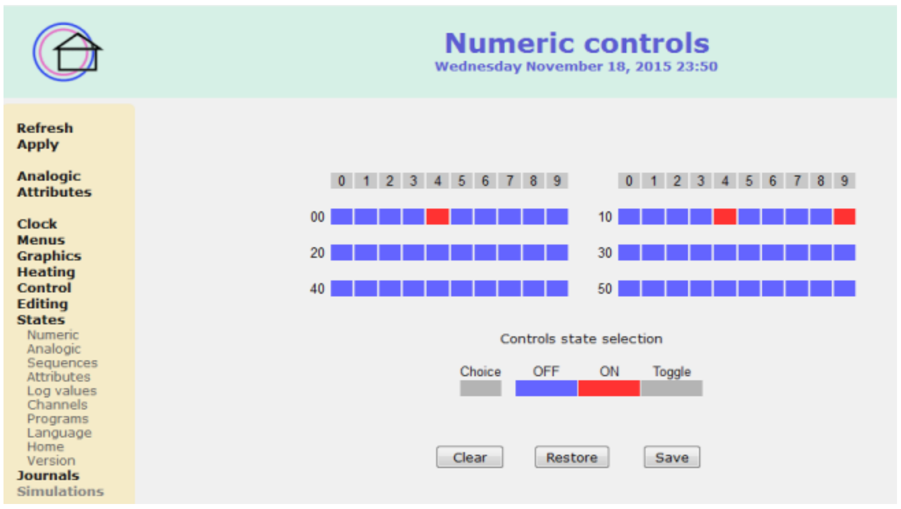
Field 0	must contain the record number of the node, except for the terminating node, which is all 0
Field 1	is the byte index in the I/O matrix
Field 2	is the BCM bank address (here 8..11, with 9 skipped, as it does not contain any usable channels)
Field 3	contains a mask with ones for input channels : none at this time
Field 4	contains a mask with ones for output channels. The mask values can be expressed either in decimal or in hexadecimal if prefixed with 0x. Note that a channel may not appear in either mask if virtual, reserved or unused, and that no channel may appear in both.

Remember that for any modification of the configuration to take effect, you must restart the application.

This configuration would create the following mapping channels → LED

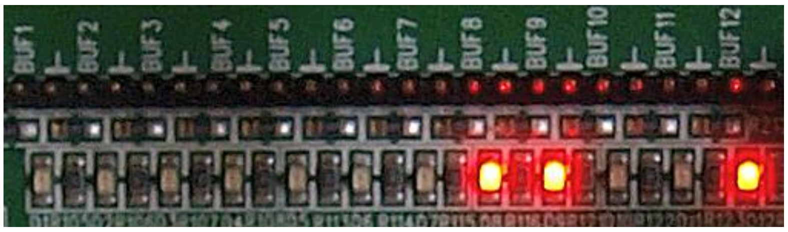
Channel	17	16	15	14	19	10	9	4
LED	B5	B6	B7	B8	B9	B10	B11	B12

For testing the correct operation of this configuration, we can use the "Numeric controls" interface and click on some of the channel cells listed above, for example 4, 14 and 19 :

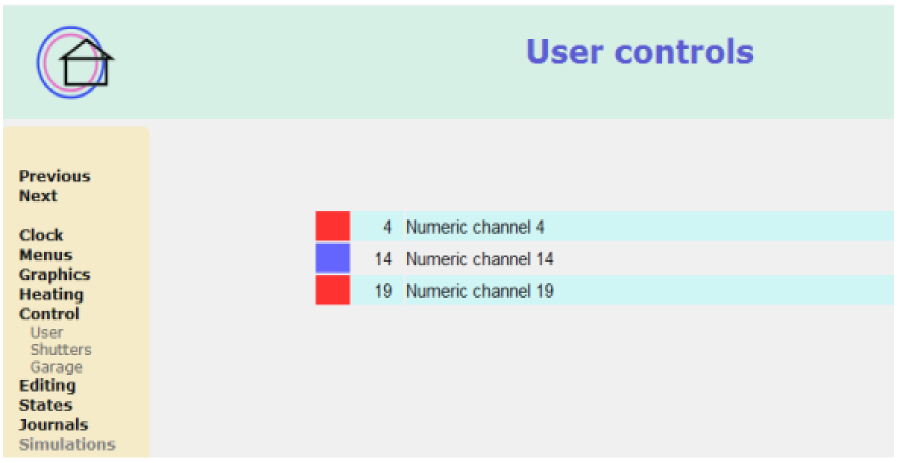


	0	1	2	3	4	5	6	7	8	9
00	🟦	🟦	🟦	🟦	🟥	🟦	🟦	🟦	🟦	🟦
10	🟦	🟦	🟦	🟦	🟥	🟦	🟦	🟦	🟦	🟥
20	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
30	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
40	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
50	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

This would switch on LED B8, B9 and B12 :



We can also switch channels with the User Control interface. Channels are added to this interface by setting the "User control" bit in the "Static bits mask" of their Parameters table. For the 3 channels above, we would get the following :



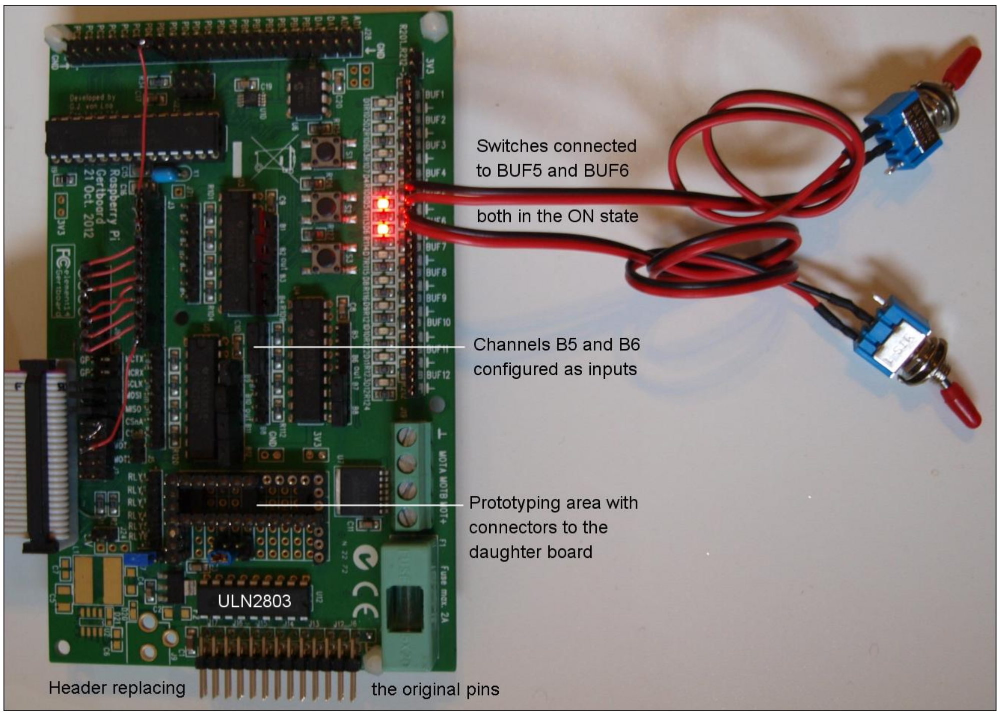
🟥	4	Numeric channel 4
🟦	14	Numeric channel 14
🟥	19	Numeric channel 19

Change the state of a LED by clicking on its colored box (blue=OFF, red=ON)

8.1.4 GPIO inputs configuration

Assume we want to use B5 and B6 as inputs. We would have to :

- on the Gertboard : move the B5 and B6 jumpers to their input position and connect switches to the BUF 5 and BUF 6 pairs of pins.


Switches connected to BUF5 and BUF6 both in the ON state
Channels B5 and B6 configured as inputs
Prototyping area with connectors to the daughter board
ULN2803
Header replacing the original pins

- in the configuration node 3, clear the bits in the output mask and set them in the input mask:

Node	Field 0	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
1	217	0	8	0	0x10	0	0
2	218	1	10	0	0xC6	0	0
3	219	2	11	**0x03**	**0x08**	0	0
4	0	0	0	0	0	0	0

- and restart the application

The effect of toggling the switches can be followed on the "Numeric controls" interface, channels 16 and 17. Note that due to the slow refreshing rate of this interface, there may be a small delay.

8.2 SPI numeric I/O configuration

8.2.1 The SPI daughter board

Due to the too small size of the prototyping area on the Gertboard, a demonstration daughter board was developed. This board features an MCP23S17 SPI I/O expander whose port A, configured as output, has its pins connected to 8 LED through an ULN2803 octuple amplifier. Its port B, configured as input, is connected to a box with 8 push buttons.
A commercial product, the PiFace expansion board, can also be used to demonstrate this fuctionality, but does not provide the GPIO and ADC functions at the same time.

8.2.2 Configuring the SPI

The following additional configuration nodes map port A to channels 24 to 31 and port B to channels 32 to 39:

Node	Port	Field 0	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
4	A	219	3	0	0	0xFF	0	0
5	B	220	4	1	0xFF	0	0xFF	0
6		0	0	0	0	0	0	0

Field 0	record number of the node
Field 1	byte index in the I/O matrix
Field 2	SPI bank address (0..7). Here the expander device address is 0, the port address within the device is 0 for port A and 1 for port B. If there are several devices, they must be allocated a continuous range of addresses starting at 0. Device address gaps are not supported.

Field 3 mask with ones for input channels. Here port B is all inputs, but it could have both.

Field 4 mask with ones for output channels. Here, due to the hardware configuration, port A is all outputs (but it also could have both).

Field 5 If necessary, outputs or inputs can be logically inverted. For example, here, on port B, we have a pull-up resistor which makes the button idle state a logical 1, which is inverted for the 8 buttons by the 0xFF value.

The byte index (field 1) may be the same in several nodes (channel concentration), provided that there is no channel collision (no multiple channels at the same bit position in the I/O matrix). The reverse (channel expansion, several times the same value in field 2) is not supported : there can be only one node for each bank address.

8.3 Analogic I/O configuration

8.3.1 Analogic raw value

An analogic *raw* or *hardware value* is the value supplied as input (or used for output) by the hardware interface, usually the output of an analogic to digital converter (or the input of a DAC). This value is stored by the system as a 16-bit value, but with a usual precision of 8 up to 12 bits (8 bits being the minimum for the specified precision of 1 percent). It is used to calculate a *physical value*, which is in turn transformed into a *presentation* or *display value*.

The currently supported ADC device is one of the Microchip MCP3xxx family. TheGertboard features an MCP3002 (10 bits, 2 channels), whose inputs are connected to the wiper pins of two 4.7K potentiometers fed by the 3.3V power supply.
The configuration is described by memory objects which are initialized by the processing of ADC descriptor nodes.
Each object supports a cluster of 1 to 16 channels occupying a contiguous window within the I/O matrix and having the same raw value acquisition method : they use the same hardware signal adapter circuit feeding one input channel of the A/D converter; each channel is connected to one input of a pre-multiplexor. Each object receives the parameters common to its supported channels from a descriptor node with the following contents:

Field 0 record number of the node

Field 1 number of the used channel of the A/D converter (0..7)

Field 2 number of the first channel in the cluster (64..127)

Field 3 size of the cluster (1..16). The number of the last channel in the cluster cannot exceed 127. Cluster overlapping (channels belonging to more than one cluster) is not supported.

Field 4 A/D device type
0 reserved for testing
1 10-bit>>2 (MCP30xx to 8-bit)
2 12-bit>>4 (MCP32xx to 8-bit)
3 10-bit raw (MCP30xx)
4 12-bit raw (MCP32xx)

Field 5 delay between input pre-mpx address write and data read (W/R), to allow the signal to settle

Field 6 filter limit (for the filter algorithm)

Field 7 list of up to 4 numeric channels switching the hardware pre-multiplexor, coded into a 32-bit value (1 channel per byte). The channel numbers range from 1 to 127, including visible and hidden channels.

For each channel number:

0 not implemented : the number of numeric channels used to generate the mpx address loses its high order bit (the number of possible multiplexor addresses is divided by 2).
Zero channels are restricted to bytes with the highest order:
0-0-10-11 is valid, 0-10-0-11 is not.
If all 4 channels are 0, there is no pre-multiplexing. This is also the case if the size of the cluster is 1.

1-127 number of the channel (bit within the I/O matrix) used to generate one bit of the mpx address. Channels 64-127 are "hidden channels", reserved for system I/O and not visible to the application (for which this range represents analogic channels)

>127 invalid

Synonym entries (i.e. declaring two or more identical channels) are prohibited.
The maximum number of addressed analogic channels is in field 3 (size of the cluster). This number may be reduced to the addressing capacity specified in the present field.
Pre-multiplexor switching is not shareable between objects, i.e. each object must have its own independent collection of switching channels.

Example:

Object	Field 0	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7
1	228	0	64	6	3	3	0	0x464241
2	229	1	80	1	3	10	0	0

The cluster of object 1 comprizes the analogic channels 64 to 69, less than its pre-mux addressing capability (8 channels). For pre-mpx, It uses numeric channels 65, 66 and 67 (hex 41, 42 and 46), which are hidden. Analogic channels 70 to 79 are unused. They could be assigned to an additional object. Channel 80 is the only analogic channel in the cluster of object 2, which has no multiplexing.

The channels used for pre-mpx must be mapped to physical outputs by one or several GPIO or SPI nodes. For object 1 in the example above, this would be :

Node	Field 0	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
2	218	8	10	0	0x46	0	0

The value 8 in field 1 indexes the hidden channel range 64 to 71 (hex 40 to 47) in the I/O matrix.

8.3.2 Analogic physical value

An analogic physical value Tp is derived from a raw value Tr by applying a channel dependent mathematical transformation to become a value in a simple relationship with some physical unit. This value is later rescaled to become a display value Td.
The rationale of this first step is to obtain a value that is consistent among all channels that produce it, by correcting the effects of different acquisition methods and characteristics dispersion.
As this value remains internal to the system, its range can be freely chosen, given that it is stored as a 16-bit unsigned value. Another consideration for choosing this range is whether this value should be compared with system or user supplied values such as setpoints or thresholds. These values being stored as unsigned bytes, some rescaling would be required before a comparison could be performed. Therefore, if a 1% precision is adequate, the range should be chosen as to fit a byte value.

The one currently defined range is the temperature, in degrees Celsius, giving the display temperature Td as a function of the physical value Tp :

Td = (0.5 * Tp) - 22.5

Td °C	Tp	Td °C	Tp	Td °C	Tp	Td °C	Tp
-22	1	-6	33	10	65	26	97
-20	5	-4	37	12	69	28	101
-18	9	-2	41	14	73	30	105
-16	13	0	45	16	77	32	109
-14	17	2	49	18	81	34	113
-12	21	4	53	20	85	36	117
-10	25	6	57	22	89	38	121
-8	29	8	61	24	93	40	125

This table shows that the chosen range allows to handle temperatures with a 0.5°C precision, consistent with what is to be expected from sensors used in home control.

The available *raw* to *physical* transformation is linear, using 3 coefficients a, b and c:

Tp = (a * Tr) / c + b

For no loss of precision, this is performed as 32-bit integer arithmetic (expecting at end a 16 or 8-bit result), meaning that Tr, Tp, a, b and c are integers, possibly rounded.

The coefficients a, b and c are stored in the channel table and editable with the channel table editor as Coefficient A, Coefficient B and Coefficient C.

As Tp must be positive, its value is possibly truncated to Tp_{min} = 0 or Tp_{max} = 65535 (or 255).

Example 1: **Tp increasing with Tr**

Assume the 10-bit ADC supplies the raw values 347 for 0°C and 883 for 30 °C :

Td °C	Tp (physical)	Tr (raw)	a/c	b
0	13	347	0,112	-25.8
30	73	883		

Tp (physical) is taken from the table above.
The theoretical value of b is rounded to the nearest integer : **b = −26**.

The value of b ranges from -127 to +127.
If negative, this value may need to be entered with the channel table editor as a 2-complement positive integer : 256−26 = 230.

Coefficients a and c are given values that produce the best possible approximation of a/c, with the conditions a < 128 and c < 256, in this case **a = 112** and **c = 100**.

Example 2: **Tp decreasing with Tr**

Assume the 10-bit ADC supplies the raw values 883 for 0°C and 347 for 30 °C :

Td °C	Tp (physical)	Tr (raw)	a/c	b
0	13	883	-0,112	111.8
30	73	347		

The theoretical value of b is rounded to the nearest integer : **b = 112**.

Coefficients a and c are given values that produce the best possible approximation of a/c, with the conditions a > −128 and c < 256, in this case **a = −112** and **c = 100**.

If negative, the value of coefficient a may need to be entered with the channel table editor as a 2-complement positive integer : 256−112 = 144. As a and b cannot be both negative, in this case, b is treated as an unsigned integer with the full range 0..255.

8.3.3 Analogic display value

The display of an analogic value with the correct numeric range and physical unit is performed by the presentation layer, which is the web page on which it appears. The required mathematical transformation (a rescaling and possibly a shift) is usually done by a JavaScript procedure executed by the web page.

There are 3 possible strategies, from simple to sophisticated:

- A hard coded transformation, including all required parameters
- A downloaded JavaScript function, including all required parameters in its code
- A hard coded algorithm valid for all channels appearing on the page and using sets of downloaded parameters

The first is simple and performing, adequate for configurations supporting a small number of different analogic values, but requires code rewriting on significant changes. The second is a good compromise in terms of initial programming effort and performance, but needs programming skills in JavaScript. The last requires the most important initial programming effort and is less perfoming due to more intensive downloading, but is the easiest to manage. All three may be used concurrently.

9 User interface development

The User Interface is made of a collection of HTML files (called *pages*) supported by CSS and JavaScript files.

9.1 The Ajax protocol

The User Interface exchanges data with the server via a special implementation of the Ajax protocol, which is supported by the `ajax.js` file, which must be included first in each `.htm` page file :

`<script src="js/ajax.js"></script>`

This file contains two JavaScript functions, one for context initialization, the other for data interchange, both called by the page supporting JavaScript file.

9.1.1 Context initialization

`function jxSetParam(baseurl, ansfunc)`

`baseurl`	URL of the action procedure on the server. Example: **`z.php`** (This is the standard name for servers that do not support PHP; for consistency, it should also be used on a PHP supporting environment)
`ansfunc`	name of a client supplied callback function that will be called once for each call of the data interchange function. This name must be used in the JavaScript supporting file of the page. It will be described later as **`ajaxAnswer()`**, but any valid function name may be used.

This function must be called before any call to the data interchange function, typically by the initialization function called when the page is loaded (this function is usually called **`iniprog()`**). It may be called more than once during the operation of the page, with different arguments, but only the context set by the last call is active.

9.1.2 Data interchange

`function jxQuery(code, val1, val2, val3)`

`code` composite argument : the concatenation of :

- a 1-letter *action verb*, for example **`r`** for read, **`w`** for write (many other action verbs exist)
- a domain letter **`@-Z`**
- an optional user ID : a user supplied alphanumeric text string starting with a letter

Example : **`rPmyid2`**

`val1` *payload* : the data to be written for a write action. Usually (but not always) empty (or ignored) for a read action. Any 8-bit character may appear in this argument, except NULL.

`val2` most frequently, the number of the record to be accessed within the specified domain. It is often a channel number. This argument is restricted to alphanumeric characters.

Special formats:

H	the current quarter hour is used as record number (0..95)
Z	only used with journalling: - on read the last written entry is returned - on write, the first free entry is written
xnnn	x : a lower case letter (a-z), nnn : a decimal number. The actual record number is the sum of *nnn* and the value of the system parameter with the id *x*.

`val3` usually the number of the field to be accessed within the record in **`val2`**, but may have another meaning depending on the action verb. This argument is also restricted to alpha-numeric characters.

If the user calls **`jxQuery()`** several times with different arguments, these requests are queued by the protocol and sent to the server in a single transaction when a mandatory terminating **`jxQuery()`**, without any argument, is called.

9.1.3 Callback function

This function processes the data returned by the server upon request of the calling page. It is therefore specific to this page and must be included and customized in the JavaScript page supporting file. Its name is set by the initialization function **`jxSetParam()`** described above. It is executed once for each non terminating call to **`jxQuery()`**. All returned arguments are text strings.

`var ajaxAnswer = function (code, val1, val2, val3)`

Arguments returned on success:

`code`	echo of the action verb of the performed operation (example: **`r`** or **`w`**)
`val1`	*payload* : the data returned for a read action. Mostly empty for a write action.
`val2`	echo of the user ID of the request (example: **`myid2`**)
`val3`	usually echo of the request **`val2`** (the record or channel number). For special formats, the actual record number is returned.

Arguments returned on error:

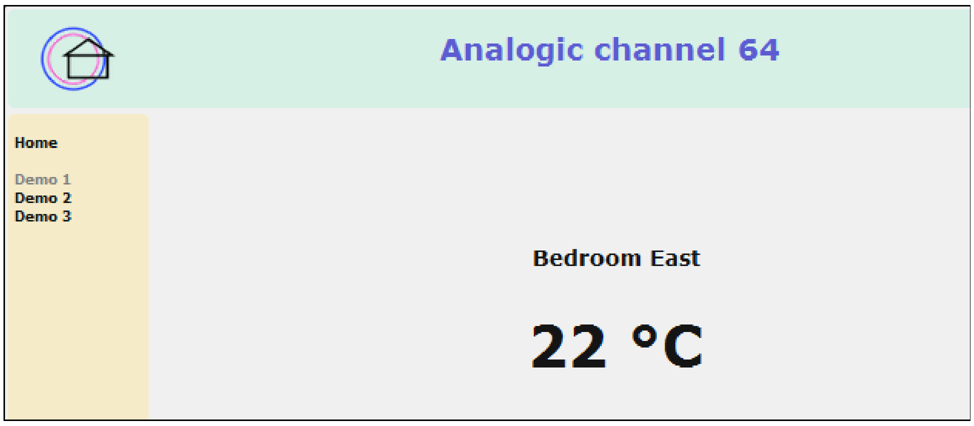
`code`	the text string `err`
`val1`	a numeric error code (the number of a message in local file `xx.php/statmsg.js`)
`val2`	echo of the user ID of the request (example: `myid2`)
`val3`	max number of records in the requested domain

9.2 Programming a page

In the following, we'll discuss the elaboration of a simple page displaying the value of an analogic channel. We'll review several versions of this page, with increasing functionality. For this exercise, we'll integrate HTML and JavaScript into a single file, but in the real world, it is better to have separate files for clarity and ease of maintenance and support (also, several pages may use the same JavaScript file).
The used page skeleton is shared by most existing pages, but, obviously, any page can be customized to the user's liking, but this aspect of development is beyond the scope of this manual.
To keep code listings within this manual as short as possible, only code snippets related to the discussion will be shown here; the reader should refer to the actual listing of the page for full context.
To keep the examples simple, most of the validity and error checking is left out.
For these examples to work, the referenced channels must have a memory table loaded and properly initialized. This can be checked by selecting the page **States** > Analogic, which displays the raw values of all analogic channels. Channels which have their memory table loaded appear with a green background.

9.2.1 Page demo_1.htm

This page displays the name of a channel and its value.



On the HTML side, things are started by:

`<body onload="iniprog();">`

The name and value of the channel are displayed in the middle of the page with the following code:

```
<h2 id="name">&nbsp;</h2>
<div class="value"><span id="val">&nbsp;</span>
<span id="unit">&deg;C</span></div>
```

This code defines 3 fields with the id : `name`, `val` and `unit`

On the JavaScript side, we select a channel (hard coded in this example) :

```
var anachan = 64; // analogic channel to display
```

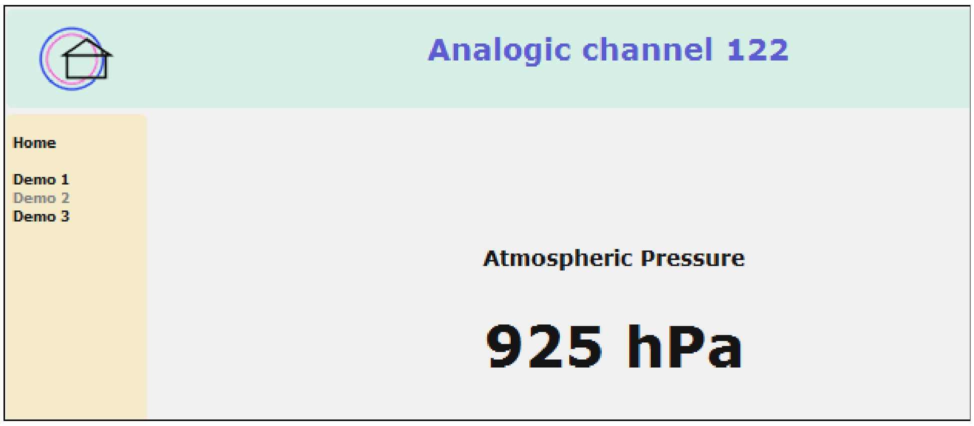
and the following 3 functions are executed, first `iniprog()`, which calls `getdata`(), which starts a transaction with the server. When the requested data is received, function `ajaxAnswer()` executes.

```
// Start data request when page is loaded
function iniprog() {
  jxSetParam('z.php', ajaxAnswer);   // init Ajax interface
  jxQuery('rOname', '', anachan, 0); // get channel name from domain O
  getdata();                         // first refresh
}

// Ask for a refresh
function getdata() {
  jxQuery('c@val', '', anachan, 5);  // get physical value of channel
  jxQuery();                         // start transaction with server
  setTimeout('getdata()', 5000);     // ask for a refresh every 5 seconds
}

// Ajax callback function
var ajaxAnswer = function (code, v1, v2, v3) {
  switch (v2) {
    case 'name':
      document.getElementById(v2).innerHTML = v1;
      break;
    case 'val':
      v1 = +v1/2 - 22.5; // convert channel physical value to temperature
      document.getElementById(v2).innerHTML = v1;
      break;
  } // switch (v2)
}
```

The **c** action verb provides read access to the memory data base rather than to a domain on disk.
v3 (usually the field number within a record) is interpreted as an identifier of some memory data associated with the channel. Here, 5 is the request to fetch the physical value of the channel. As the syntax of the **code** argument requires a domain letter, domain **@** is used as a place holder.

The **ajaxAnswer()** function is usually implemented as a **switch(code)** or a **switch(v2)**, or both in two levels, depending on how the requested data is organized. The received data is then displayed, using **v2** as the identifier of the field to fill in.

9.2.2 Page demo_2.htm



In page **demo_1**, the channel number and the physical–to-display conversion formula were hard coded, that is the same for all channels and requiring a recoding if the formula changes.
In this page we encode the channel number in the URL as following:

demo_2.htm#Xnnn

X a channel type indicator (a letter A-Z)
nnn a decimal number in the range 64..127

and we download a conversion formula as a channel related JavaScript program which is evaluated within the page. This program is called a macro and is fetched from the macro domain K. The channel type indicator **X** is translated into a record number in the macro domain (for example, assuming 4 different types, take its char code modulo 4, added to a base value, for example 150).

First, we supply 4 macros, starting at record 150 in domain K:

Type	Record	Contents
D	150	`v1=+v1/13;unit=' km/h';`
A	151	`v1=+v1/2-22.5;unit='&deg;C';`
B	152	`v1=+v1*25;unit=' hPa';`
C	153	`v1=+v1%100;unit='%';`

(Note: these macros are for demo purpose, they are unlikely to represent actual conversion formulas)

Then, we modify the code of the page to fetch and use these macros.

```
// Start data request when page is loaded
function iniprog() {
  var anaconv = convbase;          // macro record number (init to base)
  var u = (document.URL+'#').split('#');   // argument supplied by URL
  if ((u = u[1]) == '') return;            // no channel specified: give up
  anaconv += (u.charCodeAt(0) & 3);        // macro record number
  u = +u.substring(1);                     // channel number
  if ((u >= 64) && (u < 128)) anachan = u; // valid channel number present
  document.getElementById('capt').innerHTML = "Analogic channel "+anachan;
  if (!anachan) return;                    // invalid channel number

  jxSetParam('z.php', ajaxAnswer);         // init Ajax interface
  jxQuery('rOname', '', anachan, 0);       // read channel name
  jxQuery('rKconv', '', anaconv, 0);       // read conversion macro
  getdata();                               // first refresh
}

// Ask for a refresh
function getdata() {
  jxQuery('c@val', '', anachan, 5); // read value of analogic channel
  jxQuery();                        // start transaction with server
  setTimeout('getdata()', 5000);    // new read in 5 seconds
}

// Ajax callback function
var ajaxAnswer = function (code, v1, v2, v3) {
  switch (v2) {
    case 'name':
      document.getElementById(v2).innerHTML = v1;
      break;
    case 'conv':
      conv = v1;                    // macro -> global variable
      v1 = 0;                       // v1 used by macro evaluation
      eval(conv);                   // get unit name from macro
      document.getElementById('unit').innerHTML = unit;
      break;
    case 'val':
      eval(conv);                   // physical->display conversion
      v1 = Math.floor(v1*10)/10;    // keep only 1 decimal place
      document.getElementById(v2).innerHTML = v1;
  } // switch (v2)
}
```

Function **`iniprog()`** now decodes the argument supplied in the URL, which yields a channel number and a macro record number, and requests the channel name and the macro from the server.

Function **`getdata()`** remains unchanged.

The callback function uses the macro to display the name of the physical unit and to convert the channel physical value to its display value, which is right truncated to one decimal place.

9.2.3 Page demo_3.htm

In the previous demo pages, we had to handle a single channel. But what if we want to display several channels on the same page ?

First, there must be a mechanism to provide the page with the list of these channels.

- We can reuse the URL to pass this list, for example, by separating the channels with underscores:

  `demo_2.htm#A121_B122_D123_C124`

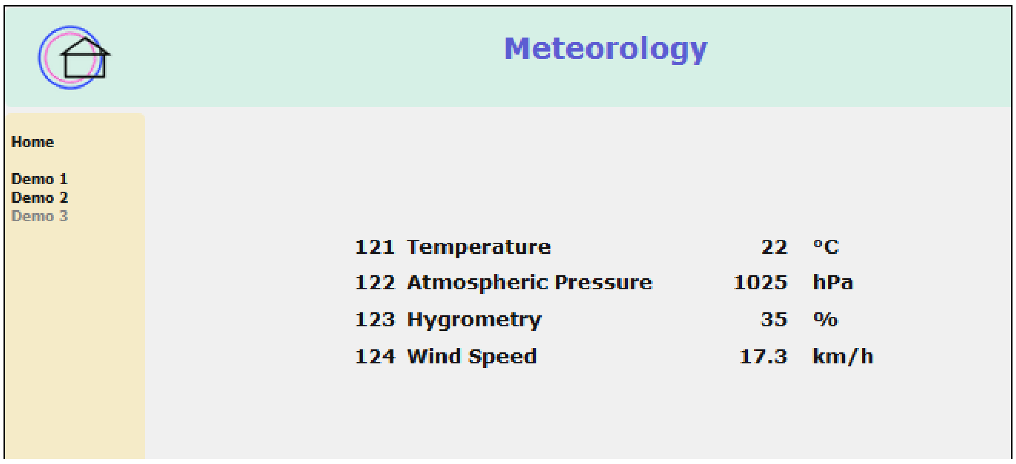
  The page would then parse this list and process each channel independently.
  Note that the same channel may appear more than once, if we want to display both its setpoint and actual value.

- We can put the same list in a record of a domain on disk. This would make modifications much easier. The names dictionary domain O has been designed for this kind of functionality. Besides storing a name for each channel, it features records with two fields, one for a list name, the other for the list itself. For example:

Record	Field 0	Field 1
161	Meteorology	**`A121_B122_C123_D124`**

To get a list, the page only needs to know its record number (which could be passed interactively by the user or by the URL, as in this example).
It could use the list name (field 0) as an overall caption for the whole display.

Assume we want to display weather data as following:



121	Temperature	22	°C
122	Atmospheric Pressure	1025	hPa
123	Hygrometry	35	%
124	Wind Speed	17.3	km/h

Four columns: channel number, channel name, current value and physical unit.
Up to 4 rows, displaying the main meteorological data, whose physical values are assumed to be available on channels 121 to 124. We reuse the macros of Demo 2 to compute the display values.

Fist, we would have to check if we have the correct channel names in domain O, records 121 to 124, field 0. Second, check if domain O, record 161 contains the caption text (Meteorology) in field 0 and the channel list in field 1. The page Demo 3 would then be loaded with the URL:

`demo_3.htm#1`

(We use the value of system parameter j as base, which is assumed to be 160).
We could reuse the same page to display completely different channels and data by just changing the hash value **`#1`**.

Putting all data on display requires several steps:

- use the hash value in the URL to get the channel list and its name. Display this name as the main caption
- split the channel list into a channel type indicator list and a channel number list and display the channel numbers in column 1.
- get the channel names from the server and display them in column 2.
- get and store locally the channel conversion macros from the server, extract and display the unit names in column 4.
- get, convert and display the channel values every 5 seconds in column 3.

The main structure in the page is a 4-row 4-column table :

```
<table border="0">
<tr><td class="number" id="c0"></td><td class="name" id="n0"></td></td>
  <td class="value" id="v0"></td><td class="unit" id="u0"></td></tr>
<tr><td class="number" id="c1"></td><td class="name" id="n1"></td></td>
  <td class="value" id="v1"></td><td class="unit" id="u1"></td></tr>
<tr><td class="number" id="c2"></td><td class="name" id="n2"></td></td>
  <td class="value" id="v2"></td><td class="unit" id="u2"></td></tr>
<tr><td class="number" id="c3"></td><td class="name" id="n3"></td></td>
  <td class="value" id="v3"></td><td class="unit" id="u3"></td></tr>
  </table>
```

In this table, every field has an ID made of a 1-letter contents type (c=channel number, n=channel name, v=display value, u=unit) and the row number, which is also the channel index (0..3). These ID are used in the requests to the server, which echoes them with the associated data, providing identification of the returned data, for immediate display or preliminary conversion.

```
function iniprog() { // Start data request when page is loaded
  var u = (document.URL+'#').split('#'); // argument supplied by URL
  if ((menu = u[1]) == '') menu = 0;      // if no menu specified, give up
  if ((+menu < 1) || (+menu > 31)) {      // check menu number validity
    show('msg', 'Invalid menu number '+menu); // fatal error message
    return;
  } // if ((+...

  jxSetParam('z.php', ajaxAnswer);      // init Ajax interface
  jxQuery('rOcapt', '', 'j'+menu, 0);   // read menu name
  jxQuery('rOs', '', 'j'+menu, 1);      // read menu value (channels string)
  jxQuery();                            // start read
}
```

After the last **`jxQuery()`,** activity is suspended until the requested data is received and the **ajaxAnswer()** function is entered. This function features a **`switch`** statement with two cases for initializations (**`s`** and **`z`**) and two cases for filling out the fields in the table (**`v`** and **`default`**).

```
var ajaxAnswer = function (code, v1, v2, v3) {
  var x = 0;     // channel loop index
  var unit = ''; // physical unit

  switch (v2.charAt(0)) {

    case 's': // v1 = string in format Axxx_Byyy_Czzz_...
      anachan = (v1+'_').split('_');  // anachan[] = Xnnn, last is empty
      var chan = +anachan[x].substr(1);
      if (chan < 64) {                // check if analogic channel
        show('msg', 'Invalid channel number '+chan); // fatal error
        return;
      }

      for (x = 0; anachan[x] != ''; x++) {
        conv[x] = (+anachan[x].charCodeAt(0)&3)+convbase; // conv[x] = rec number
        anachan[x] = anachan[x].substr(1);   // anachan[x] = channel number
        jxQuery('rOn'+x, '', anachan[x], 0); // get the channel name
        jxQuery('rKz'+x, '', conv[x], 0);    // get the conversion macro
      } // for (x=...

      xmax = x;
      jxQuery();
      break;

    case 'z': // v1 = conversion macro
      x = +v2.substr(1);        // channel index
      conv[x] = v1;             // store the conversion macro
      v1 = 0;                   // init before use by the macro
      eval(conv[x]);            // get the unit
      show('c'+x, anachan[x]); // display the channel number
      show('u'+x, unit);        // display the unit
      if (xmax > x+1) break;    // wait on last channel
      getdata();                // get actual values for all channels
      break;

    case 'v': // v1 = actual physical channel value
      x = +v2.substr(1);        // channel index
      eval(conv[x]);            // physical->display
      v1 = Math.floor(v1*10)/10; // keep only 1 decimal place
      show(v2, v1);
      break;

    default: // v1 = data for immediate display
      show(v2, v1);
      break;

  } // switch (v2...)
}
```

The initialization requires 2 steps

- **`s`** processing the channel list
- **`z`** storing the conversion macros and displaying static information. This step will trigger the next one once all channels have been processed.

The 3rd step : **`v`** converting and displaying the values of all channels is then initiated every 5th second by function **`getdata()`**.

```
function getdata() { // ask for a refresh
  for (var x = 0; x < xmax; x++) {
    jxQuery('c@v'+x, '', anachan[x], 5); // request values of anal channels
  } // for (...
  jxQuery();
  setTimeout('getdata()', 5000);
```

}

9.3 The server data access interface

Verb	Description
a	Unused
b	Read the program record associated with a channel (deprecated)
c	Read from memory data base
	0 version reporting v1 "**n.n Windows**" or "**n.n Linux**", depending on the environment n.n is the version.subversion number
	1 all numeric channels actual values v1 64-byte string, one byte per channel, Each byte : bit 0 : channel value bit 1 : channel is write protected bit 2 : channel has user control bit 3 : channel has a memory table bit 4 : channel performs a daily logging
	2 all analogic channels raw HW values as double bytes v1 128-byte string, each channel 2 bytes, big endian
	3 one numeric channel actual value as one byte • if the associated HW channel is 127, the result of a logical combination of channel values described by the MACRO record indexed in field F_program of the channel table • else, the value of the associated hardware channel (self by default). If this channel is analogic, its last position out of hysteresis is returned.
	4 one analogic channel HW value as string
	5 one analogic channel physical value as string
	6 one channel daily logging buffer (96 bytes)
	7 channel set value as string
	8 one analogic channel program value with increment
	9 one analogic channel program value as character
	10 Same as 6
	11 Status of all sequences. One byte per sequence. Contents : @ idle A active B disabled
d	Write to mem data base requiring administrator rights
	0 all numeric channels actual values passed as a 64-byte char string. The value of a channel is not changed if it is write protected.
	1 one channel : hardware value
	2 one channel : physical value
	8 clear bits in the features mask
	9 set bits in the features mask
e	Reserved for error processing
f	Unused
g	Sequence start/stop - Call a function for testing
	0 start / kill a sequence (deprecated)
	1 start a given sequence
	2 kill a given sequence
	5 fill a daily logging buffer with simu record (disabled)
	6 write a value to a given field (recnum) in all numeric channel tables(note: R_prio=1, P=space)
	7 same as 6 for all analogic channel tables
h	Read system date/time v1 day month year v2 hours :minutes :seconds v3 current quarter hour
i	Increment/decrement a program delta value If not 0, the increment is sign added to the current delta, else this delta is cleared. The new delta is written to mem and to disk.
j	Insertion in a journal file file for test v1 data to be written to the journal file v2 number of the message format v3 times to write the message

k	Set the current language
	If v2=0, return in v1 the contents of field v3, else the language to the contents of v1
l	Unused
m	Trace masks manipulation
	v1 mask table index
	v2 new mask (0=clear mask, 1=no action)
n	Simulation functions
	1 set value in channel table
	2 set channel hardware value
	3 set channel physical value
o	Unused
p	Administrator login
	v1 name:password
	Data returned :
	v1 full name
	v2 user ID
	v3 session id
q	Administrator check
	v1 session id returned by p
r	Read a record from a domain
	v2 record number
	v3 field number
	Data returned:
	v1 data in record/field
	v2 user Id
	v3 echoed record
s	Configuration data 1
	v1 number of records in domain
	v3 record size
t	Configuration data 2
	v1 number of fields in record
	v3 field size
u	Journal file params
	v1 number of last record written
	v2 number of highest writable record
	v3 number of lowest writable record
v	Read/write a channel table entry
	v1 write:payload, read:table entry index
	v2 channel number
	v3 write:table entry index, read:0
w	Write a record to a domain
	v1 payload
	v2 record number
	v3 field number
x	Expand a macro
	v2 macro record number
y	Logical combination of channel values
	v2 number of record containing channel list
	v3 number of field containing channel list
	Data returned:
	v1 logical result: 0 or 1
z	Unused
$	NOP