CasaRemote

Macro Language

I. Version Overview

CasaRemote	CasaRemote HD	CRML
V1.1		V1.0
V1.2		V1.0
V1.2.x		V1.1
V1.3.x		V1.1
V1.4		V1.2
V1.5		V1.3
V1.6		V1.5
V1.7.x	V1.0	V1.6
	V1.1.x	V1.7
V1.8	V1.2	V1.8
V1.9.x	V1.3.x	V1.9
V1.10	V1.4	V2.0
V1.10.3	V1.4.3	V2.1
	V2.0	V2.2
	V2.1	V2.3

II. Introduction

CasaRemote is featuring a macro language (CRML) that allows easy adaptation of new devices. CRML is a very simple language that is similar to shell script dialects.

Macro Integration And Execution

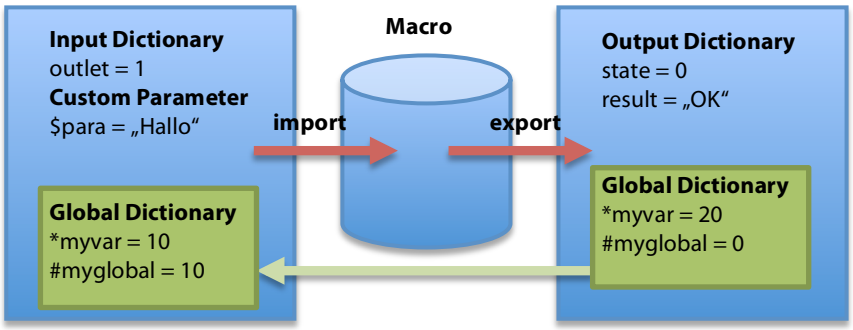
CasaRemote (CR) is using macros to abstract the differences of the physical devices. Macros can be seen as "callbacks" that are executed at various point during runtime.

CR is communicating with the CRML macros via so-called dictionaries:

- Input dictionary
  CR is setting various keys in this dictionary, which can be evaluated by the macro
- Output dictionary
  Certain macros are required to return some form of data or state. CR defines certain key that a macro has to set before finisheing execution.
- Global dictionary

A dictionary can be seen as set of environment variables in the style "*key*=*value*".
Both dictionaries can contain any variable type. A CRML macro can use the following command to read or write to the dictionaries:

- 'import' Read a value from the **input dictionary**
- 'export' Write a value to the **output dictionary**



Special keys

Macro Globals („*“-Prefix)
: Keys written to the output dictionary with key names starting with "*" are copied automatically to the input dictionary next time the macro is called and can therefore serve as global variables.

Device Globals („#“-Prefix)
: Keys written to the output dictionary with key names starting with "#" are global to all devices and can be accessed from any macro (**CRML V1.6**).

Persistent Globals („@“-Prefix)
: Keys are stored on the device an therefore say persistent even accross CasaRemote App shutdown and restart! (**CRML V2.3**).

Custom Parameters („$“-Prefix)
: Custom parameters can be defined on global device level or per sensor/socket. Custom parameters can be set by the user through the CR user interface. Not all device descriptions support custom parameters! (**CRML V1.9**).

A CasaLive device might contain any number of macros. The following paragraphs lists the macros for all devices types, their execution time and dictionary parameters.

III. Devices and their macros

Infrared Gateway

CasaRemote supports IR Transmitters that communicate via TCP/IP protocol on a given port. Both IP address and port are defined as part of the device description.

<init-macro>

Called directly after connecting with the device. Can be used to send some initialization data to the device.

Input Dictionary	
login (string)	Value of the login:password field of the device **(CRML V1.5)**

<learn-macro>

Called once the user taps on the "Learn" button in the user interface. This macro should be used to "arm" the IR Transmitter to get ready to receive a new IR code.

<learn-done-macro>

Called once a key code has been learned.

<send-macro>

Called when the user taps a IR button in the user interface.

Input Dictionary	
keycode (string)	Keycode to be send
name (string)	Name of key to be send **(CRML V1.1)**

Output Dictionary	
result (integer)	Result code: 0=Send OK, -1=Send failed

<data-macro>

This macro is called asynchronously then new data is received (typically as response to the *learn-macro*)

Input Dictionary	
fullness (integer)	Number of bytes available int the TCP/IP buffer

Output Dictionary	
keycode (string)	Received keycode. Has to be in ASCII form. For binary protocols use "bin2hex" command.
result (string)	Result message, either "OK" or a device specific error string

Controller/Aktuator

CasaRemote support IP enabled Power Outlets that can be controlled either via HTTP or low-level TCP/IP protocols.

<config-macro>

Called when the user taps the „Auto“ button in the device config menu. This macro is supposed to read the internal configuration of a device (e.g. the list of aktuators) and create appropriate outlets in CasaRemote.

Note: If this operation is complex and/or long, it is better to just kick-of the configuration reading in this macro and do the actual work in the <data-macro> asychronously.

Input Dictionary	
login (string)	Value of the login:password field of the device

$<para> (string)	Global custom parameter **(CRML V1.9)**

<init-macro>

Called directly after connecting with the device. Can be used to send some initialization data to the device.

Input Dictionary	
login (string)	Value of the login:password field of the device **(CRML V1.5)**

<get-macro>

Called once per outlet to retrieve its state when the device view is entered.

Input Dictionary	
outlet (integer)	Outlet number 0...x **(deprecated)**
ident (string)	Outlet identifier **(CRML V1.1)**
type (string)	Outlet type: switch,button,slider,toggle, colorpicker,selection,tabbar **(CRML V1.3)**

Output Dictionary	
state (integer)	Current state of outlet Type 'Switch','Toggle': 0=off , 1=on Type 'Slider' : 0 .... 10000 Type 'Selection': 0 ... x Type 'Tabbar: 0 ... 127
info (string)	Optional info string displayed next to the control, e.g. the value. Currently only used by the actor type "slider" **(CRML V1.6)**

<set-macro>

Called after the user has changed the switches in the user interface to send the new state to the device:

Input Dictionary	
outlet (integer)	Outlet number 0...x **(deprecated)**
ident (string)	Outlet identifier (ID) **(CRML V1.1)**
type (string)	Outlet type: switch,button,slider,toggle, colorpicker,selection,tabbar **(CRML V1.3)**
selections (string)	Comma separated list of selection names **(CRML V2.0)**
state (integer)	New state of outlet Type 'Switch','Toggle': 0=off , 1=on Type 'Slider' : 0 .... 10000 Type 'Selection': 0 ... x Type 'Tabbar: 0 ... 127
$<para> (string)	Global custom parameter **(CRML V1.9)**
$<para>-<id>	Custom parameter for outlet <id> **(CRML V1.9)**

(string)	

<data-macro> (CRML V1.3)

This macro is called whenever new data arrives on the device's defined IP address/port.

Input Dictionary	
fullness (integer)	Number of bytes available in the TCP/IP buffer
ids (string)	List of comma separated outlet identifiers
types (string)	List of comma separated outlet types

Output Dictionary	
state-<ID> (integer)	Optional state of outlet with <ID> Type 'Switch','Toggle': 0=off , 1=on Type 'Slider' : 0 .... 10000 Type 'Selection': 0 ... x Type 'Tabbar: 0 ... 127
info-<ID> (string)	Optional info string displayed next to the control, e.g. the value. Currently only used by the actor type "slider" **(CRML V1.6)**

Sensor

CasaRemote supports IP enabled Weather Stations that can deliver the weather data either via HTTP or low-level TCP/IP protocols.

<config-macro>

Called when the user taps the „Auto“ button in the device config menu. This macro is supposed to read the internal configuration of a device (e.g. the list of sensors) and create appropriate sensors in CasaRemote.

Note: If this operation is complex and/or long, it is better to just kick-of the configuration reading in this macro and do the actual work in the <data-macro> asychronously.

Input Dictionary	
login (string)	Value of the login:password field of the device
$<para> (string)	Custom parameter **(CRML V1.9)**

<req-macro>

Called when the device view is entered. This can be used to request the device to send new sensor value. The macro is called once per sensor.

Input Dictionary	
sensor (string)	Name of sensor to request new value
num (integer)	Sensor number
last (integer)	1=Last sensor

name (string)	Sensor name **(CRML V1.2)**
ident (string)	Sensor ID **(CRML V1.5)**
unit (string)	*metric* or *us* **(CRML V1.2)**
type (string)	Sensor type: *T=Temperature, P=Pressure, H=Humidity, W=Wind, S=Wind Speed, V=Value (Odometer style)* **(CRML V1.3)**
login (string)	Value of the login:password field of the device **(CRML V1.5)**
$<para> (string)	Global custom parameter **(CRML V1.9)**
$<para>-<num> (string)	Custom parameter for sensor <num> **(CRML V1.9)**

Output Dictionary (optional)	
<sensor name> (float)	New value for the sensor named <sensor name>

<data-macro>

This macro is called whenever new data arrives on the device's defined IP address/port.

Input Dictionary	
fullness (integer)	Number of bytes available int the TCP/IP buffer
names (string)	List of comma separated sensor names
numbers (string)	List of comma separated sensor numbers
ids (string)	List of comma separated sensor IDs **(CRML V1.5)**

Sensor values are returned via the output dictionary. For each sensor one floating point value should be exported. Only for sensors of type "Chart" a list of comma separated float values is expected:

Output Dictionary	
<sensor name> (float)	New value for gauge named <sensor name>
<sensor number> (float)	New value for gauge number <sensor number>**(CRML V1.5)**
accessByNumber (0/1)	If set to "1", sensor values are expected to be exported referenced by number instead of names **(CRML V1.5)**

IV. Basic Macro Language Syntax

All CRML commands have the following format

<command> <parameter 1> <parameter 2> ... <parameter n>

```
Example:
     setcolor 255 255 255
```

The command an all parameters are separated by one (or more) spaces. Leading or trailing spaces are ignored. All characters following the ‚%' sign are also ignored.

```
<retvar> := <command> <parameter 1> ... <parameter n>

Example:
     i := add i 1
```

Calling commands that return a value have the following format

Variables

Although CRML is a interpreted language, variables have to be declared prior to usage. Variable declaration start with the „VAR" keyword:

VAR <type> <variable 1> <variable 2> ... <variable n>

CRML supports the following variable types

- int 32-bit signed integer
- float floating point
- string array of 8-bit characters of length 256
- str*XXX* array of 8-bit characters of length *XXX*

```
Example:
     VAR int color
     VAR str64 name
```

Variables are assigned by the „:=" operator or the „set" command:

```
     i:=10
     set name "Hallo"
```

Numeric values can be given as

Decimal (e.g. 10)
Hex (e.g. 0xAB)
Float (e.g. 10.0)

Control Structures

CRML supports the following control structures:

IF Statement

if <condition>
else
endif

WHILE Loop

while <condition>
wend

REPEAT Loop

repeat
until <condition>

SELECT Statement (CRML 2.0)

select
    case <condition>
    break
    case <condition>
    break
    ...
selend

NOTE: select statements can't be nested!

Conditions must have the form
<variable> <operator> <variable or value>

Operators are standard "C" language operators.

```
Examples:

     i:=0
     while i<10
          i:=add i 1
     wend

     i:=0
     repeat
          i:=add i 1
     until i==10

     if name!="Peter"
     endif

     select
          case color==“red“
               rgb:=0xff0000
          break
          case color==“green“
               rgb:=0x00ff00
          break
          case color==“blue“
               rgb:=0x0000ff
          break
     selend
```

String Substitution

CRML supports substitution of variables within strings by putting the variable name within square braces "[]".

```
Example:
     dogs := 2
     text := "I have [dogs] dogs"

Result:
     I have 2 dogs
```

If the modifier '@' is used to subsitute an integer, its hex representation is used (**CRML V1.5**):

```
Example:
     dogs := 10
     text := "I have 0x[@dogs] dogs"

Result:
     I have 0x0A dogs
```

If instead of a variable name a decimal number is put into the braces, it is replaced by the corresponding ASCII code. **Note**: ASCII code zero and some special characters (e.g. ", [,]) can't be substituted!

```
Example:
     text := "[65][66][67]"
Result:
     "ABC"
```

ASCII Escape Codes (CRML V1.8)

CRML supports special ASCII escape codes to include arbitrary ASCII codes: "\xxx", where "xxx" represents the ASCII in decimal representation. Note that the code always has to be three digits long!

```
Example:
     text := "\065\066\067"
Result:
     "ABC"
```

Limitations

- Each CRML macro is executed in a 128Kbyte (1MB CRML 2.0) sandbox. The memory is shared between the macro and its variables. The memory is freed after the macro has finished.
- CRML does not support arithmetic expressions. Basic arithmetic has to be done by using spezial commands (see "Commands").
- CRML only supports global variables

Commands

Basic Commands

stop	Stops macro execution
res:=add a1 a2	res = a1+a2
res:=sub a1 a2	res = a1-a2
res:=mult a1 a2	res = a1*a2

res:=div a1 a2	res = a1/a2
res:=log f	res = log(f) **(CRML V1.3)**
s1:=concat s2	Append string s2 at the end of string s1
len:=strlen s	Returns the length of string s
res:=integer a1	Converts a1 (string or float) into an integer
res:=float a1	Converts a1 (string or integer) into a float
res:=htonl value	Converts an 32-bit integer from host to network byte order
res:=ntohl value	Converts an 32-bit integer from network to host byte order
res:=htons value	Converts an 16-bit integer from host to network byte order
res:=ntohs value	Converts an 16-bit integer from network to host byte order
res:=bin2hex data	Interprets the string 'data' as byte array and returns ist representation as hex string.
res:=hex2bin data	Interprets the string 'data' as array of hex values and returns ist representation as binary data.
res:=strstr str1 str2 [start]	Searches 'str2' in 'str1'. If found 'res' contains the index where 'str2' was found, otherwise -1 Optional 'start' gives the start index in 'str1' **(CRML V1.6)**

Complex Commands

---

sendhttp (DEPRECATED)

```
res:=sendhttp url [response]
 res       integer (optional return value)
 url       string
 response  string (optional)
```

Executes the specified URL 'url'. If the URL is a partial URL (not http://) the URL is automatically prefixed with http://<device-ip>/. A negative return code 'res' indicates an error, whereas '0' stands for success. The 'response' parameter is optional and contains the server response after the command returns. The command is executed synchronously and is therefore blocking!
NOTE: Please make sure the device response is not exceeding the response string size, otherwise the macros is exited with a runtime error.

---

sendhttpasync

```
res:=sendhttpasync url
 res       integer (optional return value)
 url       string
```

Executes the specified URL 'url'. If the URL is a partial URL (not http://) the URL is automatically prefixed with http://<device-ip>/. A negative return code 'res' indicates an error, whereas '0' stands for success. The command executes asynchronously. All CasaLive devices that send data have a separate, asynchronous data reception macro for this purpose.

CRML 2.0: The command returns a unique hash of the request.

---

tcpsend or tcpsend_relaxed (CRML V1.3)

```
res:=tcpsend data [response]
 res       integer (optional return value)
 data      string
 response  string (optional)
```

Sends the given string the device. The IP address and port are defined as part of the device description. The response parameter is optional and contain the device response after the command returns. A negative return code 'res' indicates an error, whereas '0' stands for success. "tcpsend_relaxed" will not automatically disconnect from the device in case of an error.
NOTE: Using the the response parameter is causeing CasaRemote to block until the device responds. This is in most cases not what you want. All CasaLive devices that send data have a separate, asynchronous data reception macro for this purpose.

tcpsend or tcpsend_relaxed (alternate form, CRML V1.5)

```
res:=tcpsend data [datalen]
 res       integer (optional return value)
 data      string
 datalen   integer (optional)
```

Sends the given string the device. The IP address and port are defined as part of the device description. The "datalen" parameter can be used to set the length of the data string (usefull if the string contains null bytes). A negative return code 'res' indicates an error, whereas '0' stands for success. "tcpsend_relaxed" will not automatically disconnect from the device in case of an error.

udpsend (CRML V1.3)

```
res:=udpsend data [port] [datalen]
 res       integer (optional return value)
 data      string
 port      integer (optional)
 datalen   integer (optional)
```

Sends the given string the device via UDP protocol. The IP address and port are defined as part of the device description. The "port" parameter is optionally used to override the standard device port. The "datalen" parameter can be used to set the length of the data string (usefull if the string contains null bytes). A negative return code 'res' indicates an error, whereas '0' stands for success.

tcpget

```
data:=tcpget count
 data      string (optional return value)
 count     integer
```

Return the last 'count' bytes received from the device. The command will not block. If less data is available, only the available data is returned.

tcppeek (CRML V1.1)

Same as as "tcpget", but leaving the data in the internal buffer. This can be used to check whether certain data is valid. Important: Make sure to use "tcpget" to actually remove the data!

extract

```
res:=extract str1 str2
 res       string
 str1      string
 str2      string
```

Searches for the string 'str2' in string 'str1' and returns the remaining characters of 'str1'.

```
Example:

     foo:="Red: 10 Green: 20 Blue: 30"
     green:=extract foo "Green: "

Result (green):

     "20 Blue: 30"
```

range

```
res:=range str1 start length
 res       string
 str1      string
 start     integer
 length    integer
```

Returns a substring of string 'str1', where 'start' is the start position (zero-based) and 'length' the number of characters.

```
Example:

     foo:="Red Green Blue"
     green:=range foo 4 5

Result (green):

     "Green"
```

char

```
res:=char str pos
 res       integer
 str       string
 pos       integer
```

Returns the character at index 'pos' (zero-based) within string 'str'.

---

setchar (CRML V1.5)

```
setchar str pos value
 str       string
 pos       integer
 value     integer (LSB only)
```

Sets the character at index 'pos' (zero-based) within string 'str' to 'value'.

---

trim (CRML V1.5)

```
trim str chars [flag]
 str       string
 chars     string
 flag      "start" or "end"
```

```
Examples:

str:=" –Hallo- "

trim str " "
Result: "-Hallo-"

trim str " -"
Result: "Hallo"

trim str " -" start
Result: "Hallo  "

trim str " -" end
Result: "  Hallo"
```

Eliminates the characters 'chars' from start and/or end of the string 'str'.

---

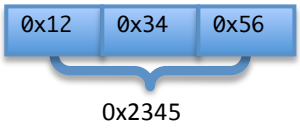
bitslice

```
res:=bitslice data startbyte startbit bits [signed]
 res            integer
 data           string
 startbyte      integer
 startbit       integer
 bits           integer
 signed         "" or "signed"
```

Interprets the string 'data' as an array of bytes and extracts a number of bits from this

array. The result is returned and optionally sign extended (if 'signed' is given as last parameter):

Example (startbyte=0, startbit=3, bits=16):



---

xpath

```
res:=xpath xml path
 res       string
 xml       string
 path      string
```

Interprets the string 'xml' as XML data and 'path' as valid XPath expression. The command evaluates the path and returns the corresponding part of the XML data.

NOTE: The XPath expression is restricted to selecting only unique, single nodes of the XML. Additionally only the selection of textual content of elements and attributes are allowed. If these rules are violated, the result will be an empty string.

CRML V1.5: It is now possible to select multiple values. They are returned as comma separated list.

---

json

```
res:=json data path error
 res       string
 data      string
 path      string
 error     integer (CRML V1.6)
```

Interprets the string 'data' as JSON data and 'path' as a *JSONPath* expression selecting a part of the JSON structure. The command evaluates the path and returns the corresponding part of the JSON data. JSON path components are separated by ".", e.g.

```
{
"sensor": [
     {
          "name": " temperature ",
          "value": 15.2,
          "unit": "°C"
          },
          {
          "name": "humidity",
          "value": 55.0,
          "unit": "%"
     } ]
}

res:=json data "sensor.[1].value"

→ res = "55.0"
```

NOTE: The JSON path expression is restricted to selecting only unique, single nodes of the JSON data. If these rules are violated, the result will be an empty string.

New in CRML V1.3:
This command is ignoring any characters before the first '{" and after the last '}'.

New in CRML V1.6:
- An optional error variable can be passed. It contains after the call "0" for OK and "-1" if a parse error occured.
- Multiple values can now be selecteb by using the "*" wildcard, e.g. "sensor.*.value". The values have to be of integral type and will be retuned as a commas separated list.
- Insteas of "[ ]" it is now possible to use "( )" to specifiy an array index, e.g. "sensor.(1).value". This is useful to avoid conflicts with the CRML variable substitution syntax.

---

import

```
res:=import key <dispose>
 res       string/integer/float
 key       string
```

Imports a value from the input dictionary. (see Macro Execution)
The addition of „dispose“ will delete the key from the dictionary after being read **(CRML 2.0).**

---

export

```
export key value
 value     string/integer/float
 key       string
```

Exports a value to the output dictionary. (see Macro Execution)

---

dictcheck

```
dictcheck key value
 value     string/integer/float
 key       string
```

Returns "1" if the key exists in the dictionary, "0" if not.

---

reconnect

```
reconnect <port>
 port      integer
```

Disconnects and immediately connects the device. A port can be given to override the port number stored within the device description. This is useful to temporary connect to a different device port. If the port number is omitted, the connection is done using the device standard port.

---

mask

```
res:=mask <value> <startbit> <numbits>
 value          integer
 startbit       integer (0-31)
 numbits        integer (0-31)
```

Masks "numbits" starting at bit "startbit" from the given value.

---

listsize

```
res:=listsize <separator> <list>
 list      string
 separator separator
```

Interprets "list" as a list separated by the character given in "separator" and returns the number of list elements. Example:

```
res:=listsize "," "Red,Green,Blue"

→ res = 3
```

listindex (CRML 2.0)

```
res:=listindex <separator> <list> <entry>
 list      string
 separator string
 entry     string
```

Interprets "list" as a list separated by the character given in "separator" and returns the first index of the „entry“ in the list. If the entry is not part oft he list „-1“ is returned.
Example:

```
res:=listindex "," "Red,Green,Blue" „Green“

→ res = 1
```

listelement

```
res:=listelement <separator> <list> <element>
 list           string (or integer or float in CRML V1.5)
 separator      string
 element        integer
```

Interprets "list" as a list separated by the character given in "separator" and returns the list element number "elements" (zero based). Example:

```
res:= listelement "," "Red,Green,Blue" 1

→ res = "Green"
```

gcdecomp (CRML V1.1)

```
res := gcdecomp <compressed> <port> <uncompressed>
 compressed     string
 port           string
 uncompressed   string
 res            integer
```

Decompresses a learned keycode from the Global Caché device GC-100 using the IR learner. If "res" is 0, the decompression was successful and the result is stored in "uncompressed".

signmsg (CRML V1.1)

```
signmsg <message>
 message   string
```

Adds a signature to the beginning of the string "message". This command is only used in conjuction with *CasaMac*.

---

delsockets (CRML V1.5)

Removes all sockets from a control device. This command will only work when called from a macro within a control device!

---

addsocket (CRMl V1.5)

```
res:=addsocket <name> <ident> <type>
 name           string
 ident          string
 type           string
```

Added a new socket to a control device. This command will only work when called from a macro within a control device! The "name" and "ident" parameters can freely defined, however the "ident" value needs to make sense in respect to the *set-state* and *get-state* macros. The following socket types are valid:

- switch — On/Off
- button — Single Button (no state)
- slider — Standard Apple slider
- toggle — Standard Apple switch
- colorpicker — RGB color picker
- selection — Selection list
- tabbar — Button row, multiple selections
- radiobar — Button row, single selection **(CRML V1.6)**
- buttonbar — Button row, no state **(CRML V1.6)**

---

addsocketselection (CRML V1.6)

```
res:=addsocketselection <name> <selection>
 name           string
 selection      string
```

This command only has an effect on socket types "selction", "tabbar, "radiobar" or "buttonbar". It adds a new selection line/button to the socket.

---

delsensors (CRML V1.5)

Removes all sensors from a ensor device. This command will only work when called from a macro within a sensor device!

addsensor(CRML V1.5)

```
res:=addsensor <name> <number> <type> <ident> [<unit>
<features> <min> <max>]
 name           string
 number         integer
 ident          string
 type           string
 unit           string (for types "C" and "L")
 features       string (for types "C" and "L")
 min            float (for types "C" and "L")
 max            float (for types "C" and "L")
```

Added a new sensor to a sensor device. This command will only work when called from a macro within a sensor device! The "name" and "ident" parameters can freely defined, however the "ident" value needs to make sense in respect to the *request-value* and *receive-data* macros. The following sensor types are valid:

- T (Temperature)
- P (Pressure)
- H (Humidity)
- W (Wind Direction)
- S (Wind Speed)
- I (Indictor)
- V (Odometer style counter)
- L (Custom Gauge)
- C (Chart)
- X (Simple Value + Unit)

For the types "Custom" and "Chart" the parameters

- unit: A unit string e.g. "°C"
- features: Sensors specific feature string (currently only "fill" for the chart type)
- min, max: Value range

have to specified.

addsensorsegment (CRML V1.5)

```
res:=addsensorsegment <num> <RGB> <start> <end> <unit>
<features> <min> <max>
 num            integer
 RGB            integer
 start          float
 end            float
```

For the sensor type "Custom Gauge" segments can be defined. Each segment is defined by

a RGB color, start and end value. Segments should be numbered starting with zero.
The RGB value is best given as Hex value in the form 0xRRGGBB.

---

today (CRML V1.6)

```
utime:=today
```

Return the current date and time expressed as elapsed seconds since 01.01.1970

---

formatdate (CRML V1.6)

```
date:=formatedate <format> <seconds>
 format         string
 seconds        integer
```

Formats the date/time "seconds" given in elapsed seconds since 01.01.1970 as a human readable string. The format string conforms to Unicode standard (see ***http://unicode.org/reports/tr35/tr35-4.html#Date_Format_Patterns***), e.g. "HH:mm"

---

bytesum (CRML V1.8)

```
sum:=bytesum <data> [len]
 data       string
 len        integer
```

Calculates the sum of all bytes in the given string

---

sendself (CRML V1.8)

```
sendself <data> [delay]
 data       string
 delay      float
```

Simulates data reception. The given data is passed to the data-macro immediately or after a given delay in seconds.

---

message (CRML V1.8)

```
message <info> [timeout]
 info       string
 timeout    float
```

Displays a message in a Head-Up dialog. The dialog disappears after the given timeout in seconds. If a timeout of 0.0 is given, the dialog has to be dismissed by the user.

httppostasync (CRML V1.8)

```
hash:=httppostasync <url> <payload> [<content-type>
<method>]
 url            string
 payload        string
 content-type   string (optional)
 method         string (optional)
```

Issues an asynchonous HTTP POST request to the given URL with the given payload data. If the URL is a partial URL (not http://) the URL is automatically prefixed with http://<device-ip>/. The optional paremeter „content-type“ can be used to set the MIME-Type of the payload.

CRML 2.0: The command returns a unique hash of the request
CRML 2.3: The optional parameter „method“ can be used to overwrite the HTTP request method, e.g. „GET“

strreplace (CRML V1.8)

```
str:=strreplace <input> <old> <new>
 input     string
 old       string
 new       string
```

Replaces all occurances of <old> in <input> with <new>.

ping (CRML V1.9)

```
resp:=strreplace <ip>
 ip        string
resp       integer
```

Sends a „Ping“ to he given IP address (or URL). Returning „1“ if the device responded, or „0“ if not.

addsensorparameter (CRML V1.9)

```
addsensorparameter <sensor-num> <name> <unit> <format>
<value>
 sensor-num     integer
 name           string
 unit           string
 format         string
 value          string
```

Adds a new custom parameter to a given sensor (sensor number). A custom parameter is defined by

- Name
- Unit
- Format
    - Integer: „i:<min>:<max>“
    - Float: „f:<min<:<max>“
    - String: „s“
    - List: „l:<element 1>|<element 2>| ...“
- Default-Value

Custom parameters can be edited through the CasaRemote user-interface and are passed to the various macros as import variables.

---

removesensorparameters (CRML V1.9)

```
removesensorparameters <sensor-num>
 sensor-num     integer
```

Removes all custom parameters for a given sensor.

---

addssocketparameter (CRML V1.9, but broken, fixed in V2.0!)

```
addsocketparameter <socket-id> <name> <unit> <format>
<value>
 socket-id      string
 name           string
 unit           string
 format         string
 value          string
```

Adds a new custom parameter to a given actuator socket (Socket ID). A custom parameter is defined by

- Name
- Unit
- Format
    - Integer: „i:<min>:<max>“
    - Float: „f:<min<:<max>“
    - String: „s“
    - List: „l:<element 1>|<element 2>| ...“
- Default-Value

Custom parameters can be edited through the CasaRemote user-interface and are passed to the various macros as import variables.

---

removesocketparameters (CRML V1.9, but broken, fixed in V2.0!)

```
removesocketparameters <socket-id>
 socket-id      string
```

Removes all custom parameters for a given actuator socket.

---

redraw (CRML 2.0)

```
redraw
```

Sends a redraw event to the UI. Cause e.g to redraw a table view.

---

xmlrpc (CRML 2.0)

```
xml:=xmlrpc <method> <para1> <para2> ...
  method   string
  paraX    string
```

Formats the given parameters as XML RPC command.

xmlrpc „setValue“ „dimmer“ 0.5

will produce the following return value

```
<?xml version="1.0"?>
<methodCall>
   <methodName>setValue</methodName>
   <params>
      <param><value><string>dimmer</string></value></param>
      <param><value><double>0.5</ double ></value></param>
   </params>
</methodCall>
```

---

normalize (CRML 2.0)

```
val:=normalize <method> <v> <min> <max>
  v      float
  min    float
  max    float
```

Normalizes the input value „v“ to a range of 0...10000 according to the following formula

```
res:=(v-min)/(max-min)*10000.0
```

---

denormalize (CRML 2.0)

```
val:=denormalize <method> <v> <min> <max>
  v      float
```

```
  min   float
  max   float
```

Denormalizes the input value „v“ in range of 0...10000 according to the following formula

```
res:=v/10000.0*(max-min)+min
```

---

httpget (CRML 2.0)

```
data:=httpget <&hash>
  hash       int*
```

Returns the most recent response of a „sendhttpasync“ or „httppostasync“ command. In addition the unique hash of the request is returned into the „hash“ parameter.

---

progress (CRML 2.0)

```
progress <info>
  info      string
```

Displays a progress dialog box:

progress open	Show dialog
progress <info>	Updates text in dialog with „info“
progress close	Hide dialog

---

deferred (CRML 2.0)

```
deferred <routine> <delay>
  routine    string
  delay      float (optional)
```

Calls the subroutine „routine“ in the devices' data macro immediately or after a given „delay“ in seconds. This can be useful to split longer tasks into smaller pieces without blocking the UI for too long.
The subroutine in the data macro is identified by the „SUB <routine-name>“ statement:

```
<data-macro>
% Normal data macro code
  ...
% Call sub-routine „myRoutine“ after 0.2 sec
  deferred „myRoutine“ 0.2
  stop
SUB myRoutine
% Sub-routine callable with ‚deferred‘
  stop
</data-macro>
```

notify (CRML 2.3)

```
notify <type> <info>
  type       string
  info       string
```

Send asynchronous notification events to CasaRemote. Currently supported

	type	**info**
Report device status	„device-status“	„OK“ or custom error string

base64encode (CRML 2.3)

```
enc:=base64encode <data>
  enc      string
  data     string
```

Encodes the data in „data“ as Base64 and returns it.

base64decode (CRML 2.3)

```
dec:=base64encode <base64>
  dec      string
  base64   string
```

Decodes the Base64 data in „base64“ and returns it.