The M32 Serial Protocol

```
Date: November 28, 2023
Version: 1.1
Authors: Willi, OE1WKL, and Christof, OE6CHD
```

The USB bus can be used for two-way communication with a connected computer. Apart from keyed or generated characters (this had been implemented already previously) the Morserino can send information about user actions (selecting menus, configuring preferences etc), or about current settings etc to the computer, and the computer can send various commands to the Morserino (which enables full control over parameters and menus).

Receiving information from Morserino

With the exception of keyed or generated characters (this can be controlled through the "Serial Output" parameter) the Morserino always sends a valid JSON object, delimited by curly braces (see https://de.wikipedia.org/wiki/JavaScript*Object*Notation for an explanation of JSON object notation).

Morserino sends a message whenever a user action is executed on the device (navigation within menus, executing a menu, navigation within parameter menu and setting parameters etc.). This can be used for displaying user actions on a larger screen, or for generating speech output as a feedback for all user actions (thus making the Morserino usable for blind and visually impaired persons).

Sending commands to Morserino

Commands sent to the Morserino start with "GET" (read values) or "PUT" (write values) and additional parameters. Commands are ended by a single carriage return (\n or ascii 10). Responses to GET commands are of course JSON objects, while as a rule there are no responses to PUT commands.

Syntax of commands

GET *object*

GET *object*/*specifier*

PUT *object*/*specifier*/*value*

GET and PUT are separated by a blank from the following parts of the command, while object, specifier and value are separated by a slash (/).

GET, PUT, *object* and *specifier* can be upper or lower case, but value is case sensitive.

Enabling and disabling the M32 Protocol

After establishing the connection physically, the M32 protocol needs to be enabled on the Morserino. This is done with the following command:

`PUT device/protocol/on`

As a confirmation device information is returned, e.g.:
`{"device":{"hardware":"2nd edition","firmware":"5.0","protocol":"1.0"}}`

From this response the connected computer program gets not only confirmation that the communication has been established, but also information about the hardware used, as well as the firmware and protocol versions.

While the protocol is enabled, the time-out of the connected Morserino is disabled, i.e. it will not go into sleep modus.

The protocol can also be disabled with the command: `PUT device/protocol/off`

Success and Error Feedback

When the M32 could parse and execute the command, an "ok" object is returned, e.g.

`{"ok":{"content":"OK"}}.

Exceptions are the `PUT control` commands: they return the "control" objects, in the same way as `GET control`.

When invalid commands are sent to the Morserino, an "error" object is returned, e.g.

`{"error":{"name":"INVALID Value xxx"}}` .

JSON objects sent by Morserino as a result of user activities

"menu"

As a result of menu navigation, a *menu* object is sent with the following properties:

- "content" (type String): The complete menu path, with elements separated by slash (/).
- "menu number" (type Number): A number to identify this menu item.

- "executable" (type Boolean): If false, there are sub-menu items, if true this can be executed.
- "active" (type Boolean): Always false when resulting from user activity.

Example:

```
{"menu":{"content":"CW Generator/..",
"menu number":2,"executable":false,"active":false}}
```

"control"

As a result of changing the speed of the keyer, or the audio volume, a control object is sent with the following properties:

- "name" (type String): "speed" or "volume".
- "value" (type Number): For "speed": keyer speed in words per minute, for "volume" a number between 0 and 19.

Example:

```
`{"control":{"name":"speed","value":16}}`
```

"config"

As a result of navigating the parameters menu, a config object is sent with the following properties:

- "name" (type String): The name of the currently selected parameter.
- "value" (type Number): The current value, as stored internally, as a number.
- "displayed" (type String): How the value is displayed (often as a meaningful text).

Example:

```
{"config":{"name":"External Pol.","value":0,"displayed":"Normal"}}
```

"activate"

As a result of activating a menu, an activate object is sent with the following property:

- "state" (type String): can be any of "EXIT", "ON", "SET", "CANCELLED", "RECALLED", "CLEARED"

Example:

```
{"activate":{"state":"ON"}}
```

"message"

Sometimes, as a result of some user action, a message is displayed on the screen. In these cases a message object is sent with the following property:

- "content" (type String): The message body.

Example:

```
{"message":{"content":"Generator Start / Stop press Paddle  "}}
```

Objects and their GET and PUT commands

Device Information

`GET device` Returns the properties „hardware“ (can currently be "1st edition“, or “2nd edition“) „firmware“ (the firmware version number, and „protocol“ (the M32 Protocol version).

Example:

```
{"device":{"hardware":"2nd edition","firmware":"5.0","protocol":"1.0"}}
```

`PUT device/protocol/on`
This switches the M32 protocol on (you will get device information back; you will also get device info when command is sent while protocol is already ON); while this is on, there will be no time-outs, whatever the parameter says!

`PUT device/protocol/off`
This switches the M32 Protocol off (time-out as defined in its parameter will be in effect again).

Control Speed and Volume

`GET control/speed`

This returns the properties "name", "value", "minimum" and "maximum" for keyer speed (in words per minute - wpm.

`GET control/volume`

This returns the properties "name", "value", "minimum" and "maximum" for the volume control.

`PUT control/speed/<value>`

Use this to set the keyer speed to (numeric value, words per minute).

`PUT control/volume/<value>`

Use this to set the speaker volume to (numeric value, 0 = no sound, 19 = maximum volume).

The Menu

`GET menus`

This returns a list of all menu items, with the properties "content" (a string giving the complete menu path), "menu number" (a unique number for each menu item), and „executable“ (a boolean value that indicates if this can be executed - if "true" - or it has sub-menu items - if "false").

Example:

```
{
"menus": [
    {
        "content": "CW Keyer",
        "menu number": 1,
        "executable": true
    },
    {
        "content": "CW Generator/..",
        "menu number": 2,
        "executable": false
    },
    {
        "content": "CW Generator/Random",
        "menu number": 3,
        "executable": true
    }, ...
 ]}
```

`GET menu`

Returns the current menu object with properties "content" (= menu path), "menu number", "executable" and "active"; if called while a mode (eg CW Keyer) is running, (and not just selected in the menu), the property "active" shows "true", otherweise "false".

Example:

```
{"menu":{"content":"CW Generator/English Words",
"menu number":5,"executable":true,"active":true}}
```

`PUT menu/set/<number>`

Set the main menu to the requested menu entry (even if it is not executable!). You have to use the menu number.

`PUT menu/start`

If in menu mode, start the currently selected menu, if it is executable (do nothing when not in menu mode, or when the current menu entry is not executable).

`PUT menu/start/<menu number>`

Start the command that has the number `<menu number>` (only if it is executable!).

For this, the M32 must be in the main menu (if not sure, execute put menu/stop before doing this, or check with `GET menu` if the "active" property is "false")

`PUT menu/start now`

`PUT menu/start now/<menu number>`

These commands perform basically the same task as `PUT menu/start` and `PUT menu/start/<menu number>` ; there is a difference only when starting one of the CW Generator modes: normally these modes, after starting them, wait for a paddle (or key) action to get going. Startimng them with any these commands will get them going immediately, without waiting that the user presses a key or touches a paddle.

`PUT menu/stop`

If M32 is active or in the preferences menu, stop it and go to main menu.

Configuration (Parameters)

`GET configs`

This returns a list of all configurable parameters, with the properties "name" (= parameter name), "value" (as numerical values), and "displayed" (how the value has to be displayed).

Example:

```
{"configs":[{"name":"Encoder Click","value":1,"displayed":"On"},
{"name":"Tone Pitch","value":10,"displayed":"622 Hz e2"},
{"name":"External Pol.","value":0,"displayed":"Normal"},...
]}
```

`GET config/<parameter name>`

This returns all details of that parameter (or an error if an invalid parameter name has been given; upper

and lower case in parameter names are not significant). The returned properties are:

- "name" (type String): the name of the parameter,
- "value" (type Number): the current value of the parameter, "description" (type String): a more verbuous description what the parameter is about,
- "minimum" (type Number),
- "maximum" (type Number),
- "step“ (type Number),
- "isMapped“ (type Boolean): "true" if the value has to be shown as some character string, and
- "mapped values" (type Array of Strings): for the value to text mappings, if applicable

Example:

```
{"config":{"name":"Keyer Mode","value":2,
"description":"Iambic Modes, Non-squeeze mode, Straight Key mode",
"minimum":1,"maximum":5,"step":1,"isMapped":true,
"mapped values":["","Iambic A","Iambic B","Ultimatic",
"Non-Squeeze","Straight Key"]}}
```

`PUT config/<parameter name>/<value>`

This sets the named parameter to the specified value - the value must be the NUMERIC value, and not the textual representation!

Example to set Keyer Mode to Iambic B:

```
PUT config/keyer mode/2
```

Snapshots

`GET snapshots`

This command returns a list of all currently stored snapshots.

Example:

```
{"snapshots":{"existing snapshots":[1,2,3]}}
```

`PUT snapshot/store/<n>`

Store the current parameters (and the currently selected menu item) in snapshot n (n = 1..8).

`PUT snapshot/recall/<n>`

Recall parameters (and menu selection) from snapshot n.

`PUT snapshot/clear/<n>`

Clear (i.e., delete) snapshot n (n = 1..8).

Stored Text File

`GET file/size`

This command return some usage statistics of the SPIFFS file system (used for the file the file player can play), with properties "size" (type Number; the size of the current player file in bytes) and "free" (type Number; the size of the available storage in bytes. Be aware that the actual free space might be a little bit less than indicated, because file space is allocated in chunks).

Example:

```
{"file":{"size":21,"free":1498219}}
```

`GET file/text`

This command returns the contents of the text file used by the file player, in an object named "file", as a property (type String) called "text".

Example:

```
{"file":{"text":"The first line of a small text file.\n
The second line, with a pause at the end. <p>\n"}}
```

`GET file/first line`

This returns the contents of the first line of the text file used by the file player, in an object named "file", as a property (type String) called "first line".

If you use the comment feature for text files (a line beginning with "" or "/c" is regarded as a comment and not played), you can use this to give you some indication about the file contents.

Example:

```
{"file":{"first line":"<c> This file contains sample QSO text."}}
```

`GET file` This is just a short version of `GET file/first line`.

`PUT file/new/<line of text>`

This command replaces an existing existing file with a new file, and place a line of text into the new file.

`PUT file/append/<line of text>`

This will append a line of text to the existing file. By using `PUT file/new/<line of text>` and then a series of `PUT file/append/<line of text>` commands you can upload a text file with many lines of text.

WiFi Configuration

`GET wifi`

This command will return the currently defined wifi entries (the properties "ssid" and "trxpeer", but NOT "password" (!), and if the property "selected" (type Boolean) is true, this entry will be used selected for connection.

Example:

```
{"wifi":[{"ssid":"Shackrouter","trxpeer":"cq.morserino.info","selected":false},
{"ssid":"MyHomeRouter","trxpeer":"cq.morserino.info","selected":true},
{"ssid":"","trxpeer":"","selected":false}]}
```

`PUT wifi/ssid/<n>/<ssid>`

This sets the SSID for WiFi setting (n = 1..3).

`PUT wifi/password/<n>/<password>`

This sets password for WiFi setting .

`PUT wifi/trxpeer/<n>/<trxpeer>`

This sets the entry for TRX Peer for WiFi setting .

`PUT wifi/select/<n>`

This selects one of the three WiFi entries as the one to be used for connecting.

Koch Lessons

`GET kochlesson`

This returns the currently selected Koch lesson, with properties "value" (type Number; the numeric value of the currently set Koch lesson), "minimum" (type Number), "maximum" (type Number), and "characters" (type Array of Strings; thee are the characters in the currently selected order of characters (parameter "Koch Sequence").

Example:

```
{"kochlesson":{"value":9,"minimum":1,"maximum":51,"characters":
["m","k","r","s","u","a","p","t","l","o","w","i",".","n","j","e","f",
"0","y","v",",","g","5","/","q","9","z","h","3","8","b","?","4","2",
"7","c","1","d","6","x","-","=","<sk>","+","<as>","<kn>","<ka>",
"<ve>","<bk>","@",":"]}}
```


`PUT kochlesson/<n>`

This sets Koch lesson to lesson number .

Automated CW Keyer and Memory Keyer

`PUT cw/play/<text to be played>`

This command plays the text string in CW (like a memory keyer - needs to be in CW Keyer mode or in Morse Trx mode, cannot work in LoRa or WiFi Trx modes). Keying will be stopped as soon as something is being keyed manually, or with the `PUT cw/stop` command.

`PUT cw/repeat/<text to be played>`

This is similar to the command above, but the text string is repeated until stopped.

`PUT cw/stop`

This will stop the CW player (it has the same effect as beginning to key manually while the commands `PUT cw/play` or `PUT cw/repeat` are active).

Note: you can use "<p>" or "\p" within `<text to be played>` to generate a short pause.

`PUT cw/store/<n>/<content>`

Store `<content>` in permanent memory number `<n>` (n is 1 .. 8); if `<content>` is an empty string, this memory is deleted. `<content>` can be normal Morse code characters, pro signs, e.g. "<bk>", and also "[p]" or "\p" for a pause.

`PUT cw/recall/<n>`

Generate Morse code from the content in memory number `<n>` ; if `<n>` is 1 or 2, do this until stopped by touching a paddle, or by the `PUT cw/stop` command.

`GET cw/memories`

Get a list of memory numbers that have some content stored.

Example:

```
get cw/memories
-> {„CW Memories":{"cw memories in use":[1,3,4]}}
```

`GET cw/memory/<n>`

Get contents of memory number `<n>`.

Example:

```
get cw/memory/
-> {"CW Memory":{"number":1,"content":"cq cq cq de oe1wkl oe1wkl [p]"}}
```

Note: Content stored in permanent memory can be recalled ("played") in Morserino-32's keyer mode, even when the Morserino is operated stand-alone and there is no serial connection to a computer (see user manual for details).