TUC 2 User Guide

for metraTec TUC 2 Modules and derived Products

Date: April 2019

Version: 2.0-RC1
For Firmware Version: 02.12
User Edition

Table of Contents

Typographic Conventions

Special typographic conventions and highlightings are used in metraTec protocol guides and other documents to streamline content that would otherwise be hard to express (e.g. syntax descriptions) and in order to provide a consistent look across metraTec documentation.

The following table summarizes typographic conventions and their descriptions:

Convention	Description
`COMMAND`	A command name, i.e. the *literal* name of a command in a metraTec protocol. For instance, **RST** would correspond to the literal characters of a command that could be sent to a metraTec device.
`Literal`	Highlights a *literal value* directly representing the literal characters that have to be used (e.g. in a protocol). For instance `UC0` would correspond to the literal characters as they could be returned by a metraTec device.
`Token`	This convention highlights a replaceable (abstract) *`Token`* that in contrast to a literal token is a placeholder for some other value that must be substituted by the user. The abstract *`Token`* is usually documented in more detail.
<LITERAL>	Represents a literal character that cannot be printed as such or needs to be highlighted specifically and is therefore formatted as an abstract identifier. Examples include "*<CR>*" — representing the carriage return character (ASCII 13) — and "*<SPACE>*" — representing one or more space characters (ASCII 32). This special formatting is used both in syntax descriptions, command and response examples.
`{` *`Construct`* `}`	This convention highlights that a *`Construct`* is required. It is most commonly used in syntax descriptions to highlight that a parameter *must* be specified in the position that this construct is used.
`[` *`Construct`* `]`	Highlights that a *`Construct`* is optional. It is most commonly used in syntax descriptions to highlight that a parameter *may* be specified in the position that this construct is used.
`Construct` `...`	Highlights that a *`Construct`* may be repeated many times.
…Name…	The horizontal ellipsis "…" may be used in syntax descriptions to represent arbitrary characters. The arbitrary character field may be given a *`Name`* in order to document it in more detail.
$Alternative_1$ `|` $Alternative_2$ `...` `|` $Alternative_n$	Highlights that in the position of this construct one of n alternatives may be used.
`Literal Line 1` *`Literal Line 2`* *`Literal Line 3`*	A literal block of text. It is often used to document example protocol exchanges or code examples. In the former case, literal placeholders like "*<CR>*" may be included in the code block to express that lines are separated by carriage return. In the latter case, programming language source

Convention	Description
	code may be syntax highlighted. These literal blocks of code may also contain callout graphics or line annotations to document each line of text.
» `Command` « `Response`	In examples of a command-response exchange, the literal examples of the ***`Command`*** and ***`Response`*** may be highlighted differently. Otherwise these literal blocks of text are formatted the same as described above.
 **Note** Paragraph	A paragraph set off from the text to highlight noteworthy information.
 **Warning** Paragraph	A paragraph set off from the text to highlight information necessary to prevent harm to electronic devices or persons.

Chapter 1. Config-Shell Commands

This list gives an overview of all available instructions to configure the TUC 2 via its TCP-based configuration shell. The commands often have several possible answers, some of which indicate an error.

Command	Name	Description
PING	Ping Device	Pings the device.
GMC	Get MAC Address	Returns the device's MAC address.
RSN	Read Serial Number	Returns the serial number.
REV	Get Revision	Returns the firmware name, hardware requirement and firmware version.
RFW	Read Firmware	Returns the firmware name and version.
RHR	Read Hardware Revision	Returns the hardware version.
RHN	Read Hardware Name	Returns the hardware name.
RHW	Read Hardware	Returns hardware name and version.
BTLREV	Bootloader revision	Returns the bootloader name and version.
SNAME	Set Module Name	Sets the module name.
GNAME	Get Module Name	Returns the module name.
SIP	Set IPv4 Addressing	Configures the IPv4 addressing.
GIP	Get IP Address	Get the current IPv4 addressing settings.
SUART	Configure UART Line Settings	**SUART** allows to configure various parameters of the specified UART interface. **Note** There also is a **UART** alias for TUC 1 backwards compatibility. The only difference to the TUC 1 version is that **HARDWARE** is not yet supported on the TUC 2.
GUART	Get UART Line Settings	Returns the UART configuration.
SMODE	Set UART mode	Sets the mode of a specified UART.
GMODE	Get UART mode	Gets the mode of a specified UART.
SCLIENTRIV	Set Client Reconnect Interval	Sets the reconnect interval in client mode.
GCLIENTRIV	Get Client Reconnect Interval	Returns the client reconnect interval.
SKEEPALIVE	Configure TCP Keep Alive Probes	Configure TCP Keep Alive probes.
GKEEPALIVE	Get TCP Keep Alive Settings	Returns the TCP Keep Alive settings.
SFLBYTE	Set Flush Byte	Sets or disables the flush byte.
GFLBYTE	Get Flush-Byte	Gets the flush-byte.
SSENDTT	Set Send Timeout	Sets the minimum time between receiving a byte on the specified UART and sending the buffered data via the associated TCP connection. If the timeout value is set to any value

Command	Name	Description
		greater than 0, every byte received will restart a timeout of the configured interval, delaying the output of all buffered data bytes to the TCP connection. The buffered data will finally be output if: 1. The timeout expired without a byte being received by the UART 2. When a flush-byte is received, in case this feature is activated (refer to **SFLBYTE**) 3. The internal buffer reaches the maximum TCP payload size (normally 1460 bytes) Setting the timeout to 0 deactivates the delay and received bytes are sent as soon as possible. Nevertheless a TCP packet may still contain multiple bytes. **Note** For compatibility with the TUC 1, there is a **SENDTT** alias.
GSENDTT	Get Send Timeout	Gets the send timeout.
SNMP	Configure SNMP	Sets/Gets various SNMP settings.
SDISCOVERY	Set Discovery Protocols	Sets the discovery protocols to use.
GDISCOVERY	Get Discovery Protocols	Gets the enabled device discovery protocols.
APPLY	Apply Settings	Applies staged settings.
DISCARD	Discard Settings	Discards staged settings.
FACTORY	Restore Factory Settings	Restores factory settings.
RESET	Reset Device	Resets the device.
CLOSE	Close Connection	Closes the connection.

Table 1. Overview of Config-Shell Commands

1.1. Ping Device (PING)

Tests the connection to the device.

Instruction

PING *<CR>*

Examples

```
PING<CR>
```

Example 1. Ping the device, expect OK! answer

Return Values in Case of Success

`OK! <CR>`

Return Values in Case of Failure

"ERR <CR>" or "UPA <CR>"

1.2. Get MAC Address (GMC)

Returns the MAC address of the device. This address was assigned during production and is unique for every TUC 2 device. It may be useful when requesting product support.

» `GMC<CR>`

« `OK! 00:50:C2:DA:2F:66<CR>`

*Example 2. **GMC** command and answer*

Instruction

GMC <CR>

Return Values in Case of Success

OK! {MAC-Address}
The MAC address is always formatted as six hex-encoded bytes, seperated by colons.

Return Values in Case of Failure

"ERR <CR>" or "UPA <CR>"

1.3. Read Serial Number (RSN)

Returns the serial number of the device which is a unique identifier assigned during production. It might be useful when requesting product support.

» `RSN<CR>`

« 2019010203040000<CR>

*Example 3. **RSN** command and answer*

Instruction

RSN *<CR>*

Return Values in Case of Success

{Serial}
16-byte ASCII string, encoding a timestamp of the format: `YYYYMMDDhhmmssrr` (whereas `r` represents reserved bytes).

Return Values in Case of Failure

`NOTSET` *<CR>*
The serial number has not yet been programmed. This should not happen. Should this error nevertheless be encountered, please get in contact with metraTec support.

"ERR *<CR>*" or "UPA *<CR>*"

1.4. Get Revision (REV)

On the revision command the device returns the name of the currently installed firmware, the minimal hardware version that is required to run this specific revision of the firmware and the actual version number.

» **REV***<CR>*

« TUC 02030212*<CR>*

*Example 4. **REV** command and answer*

Instruction

REV *<CR>*

Return Values in Case of Success

{Name}{HwReqVersion}{FwRevision}
16 bytes firmware name (right-padded with spaces), followed by 4 bytes hardware-requirement version, followed by 4 bytes firmware-revision.

Return Values in Case of Failure

"ERR *<CR>*" or "UPA *<CR>*"

1.5. Read Firmware (RFW)

This command returns the name and version of the currently installed firmware.

» **RFW**<*CR*>

« TUC 0212<*CR*>

*Example 5. **RFW** command and answer*

Instruction

RFW <*CR*>

Return Values in Case of Success

{Name}{Revision}
16 bytes firmware name (padded with spaces), followed by 4 bytes hardware revision.

Return Values in Case of Failure

"ERR <*CR*>" or "UPA <*CR*>"

1.6. Read Hardware Revision (RHR)

This command returns the hardware revision of the TUC 2 which corresponds to the PCB layout version printed on the board. This number consists of four ASCII characters in the form MMSS (2 bytes major version, 2 bytes minor version). It might be required when requesting product support.

Note

This command is deprecated, although still supported by the TUC 2. Use **RHW** instead.

» **RHR**<*CR*>

« 0203<*CR*>

*Example 6. **RHR** command and answer*

Instruction

RHR <*CR*>

Return Values in Case of Success

{Revision}
4 bytes hardware revision.

Return Values in Case of Failure

"ERR <CR>" or "UPA <CR>"

1.7. Read Hardware Name (RHN)

This command returns the hardware name of the TUC 2 which corresponds to the PCB layout name printed on the board. This name consists of up to sixteen ASCII characters. It might be required when requesting product support.

Note

This command is deprecated, although still supported by the TUC 2. Use **RHW** instead.

» **RHN**<CR>

« TUC <CR>

*Example 7. **RHN** command and answer*

Instruction

RHN <CR>

Return Values in Case of Success

{Name}
16 bytes hardware name (filled with spaces)

Return Values in Case of Failure

"ERR <CR>" or "UPA <CR>"

1.8. Read Hardware (RHW)

Returns the name of the hardware and its version. These are identical to the print on the circuit board.

» **RHW**<CR>

« TUC 0203<CR>

*Example 8. **RHW** command and answer*

Instruction

RHW *<CR>*

Return Values in Case of Success

{Name}{Revision}
16 bytes hardware name (padded with spaces), followed by 4 bytes hardware version.

Return Values in Case of Failure

"ERR *<CR>*" or "UPA *<CR>*"

1.9. Bootloader revision (BTLREV)

BTLREV returns the name of the currently installed bootloader and its version number. This may be useful when requesting product support.

» **BTLREV***<CR>*

« BTL_TUC_DEBUG 0104*<CR>*

*Example 9. **BTLREV** command and answer*

Instruction

BTLREV *<CR>*

Return Values in Case of Success

{Name}{Revision}
16 bytes bootloader name, followed by 4 bytes version number

Return Values in Case of Failure

"ERR *<CR>*" or "UPA *<CR>*"

1.10. Set Module Name (SNAME)

The **SNAME** command allows configuring an up to 54 character-wide human readable module name. These module names are reported by the TUC 2 during device discovery via UPnP or the "Legacy Discovery Protocol" (see **SDISCOVERY** command).

Note

This command is available under **NAME** alias for backwards compatibility with TUC 1 firmwares. It is worth noting that the TUC 1 would collapse whitespace in the module name and supported only 39 character-wide module names.

Instruction

SNAME *<SPACE>* {*…Module name…*} *<CR>*

Parameters

Name	Type	Description
Module name	Any String	All characters between *<SPACE>* and *<CR>* will be interpreted as the new module name.

Examples

```
SNAME metraTec TUC 2 Module<CR>
```

Example 10. Set the module name

```
NAME metraTec TUC Module<CR>
```

Example 11. Set the module name (legacy alias)

Return Values in Case of Success

OK! *<CR>*

Return Values in Case of Failure

UPA *<CR>*
: Module name is longer than 54 characters.

"ERR *<CR>*" or "UPA *<CR>*"

1.11. Get Module Name (GNAME)

Returns the module name configured by the **SNAME** command. In other words, this returns staged configuration, not necessarily the device's running configuration.

```
» GNAME<CR>
« OK! metraTec TUC 2 Module<CR>
```

*Example 12. **GNAME** command and answer*

Instruction

GNAME *<CR>*

Return Values in Case of Success

OK! {Module Name}
: All characters after `OK!` and the following *<SPACE>* character up to the trailing *<CR>* represent the currently configured module name.

Return Values in Case of Failure

"ERR *<CR>*" or "UPA *<CR>*"

1.12. Set IPv4 Addressing (SIP)

The **SIP** (or **SIP4**) command allows to configure the acquisition method of the IP address (static or dynamic). For static adressing, the device IP address, its subnet mask and the default gateway IP addresses may be configured.

Note

This command is largely TUC 1 compatible, but allows configuring a separate AutoIP addressing mode and allows gateway IPs to be omitted.

1.12.1. Set IP acquisition method to DHCP (DHCP)

Sets the IP acquisition method to DHCP. In this mode the IP-address, subnet-mask and gateway address will be supplied by a local DHCP server.

If acquiring an address via DHCP fails, the device will aquire an IP via AutoIP (see RFC3927 [https://tools.ietf.org/html/rfc3927]) while retrying DHCP. This ensures that the device will always have some kind of IP address and can be discovered with the TUC Config Manager.

Instruction

SIP *<SPACE>* DHCP *<CR>*

Examples

```
SIP DHCP<CR>
```

Example 13. Set IP-mode to DHCP

1.12.2. Set IP acquisition method to AutoIP (AUTOIP)

Sets the IP acquisition method to AutoIP. The device will pick an unused IP-address in the range from `169.254.1.0` to `169.254.255.255` on its own. Refer to RFC3927 [https://tools.ietf.org/html/rfc3927] for details on this process.

In DHCP addressing mode, the device will also aquire an AutoIP soon after it fails to aquire one via DHCP.

Instruction

SIP *<SPACE>* AUTOIP *<CR>*

Examples

```
SIP AUTOIP<CR>
```

Example 14. Set IP-mode to autoconf

1.12.3. Configure IPs statically

Uses statically configured IP addresses.

Instruction

SIP *<SPACE>* {*…IP, Netmask and Gateway…*} *<CR>*

Parameters

Name	Type	Description
IP, Netmask and Gateway	Any String	Up to three IP-addresses in dotted-decimal-notation (refer to the example above) can be supplied, separated by *<SPACE>*. The first address represents the device IP-address, the second one the subnet mask and the third one the gateway address. The gateway IP address may be ommitted, which is equivalent to setting a `0.0.0.0` gateway.

Examples

```
SIP STATIC 192.168.2.239 255.255.255.0 192.168.2.1<CR>
```

Example 15. Set static IP address

Return Values in Case of Success

OK! *<CR>*

Return Values in Case of Failure

ERR *<CR>*
: This error may indicate that invalid or semantically pointless IP addresses have been supplied (eg. netmask cannot be translated to a prefix length).

"ERR *<CR>*" or "UPA *<CR>*"

1.13. Get IP Address (GIP)

GIP (or **GIP4**) returns the current IP-address settings of the device. In case of dynamic addresses, the current acquisition method (DHCP or AUTOIP) will be returned. If a static address is set up, it will be returned instead, along with the subnet-mask and the gateway-address.

» **GIP**<*CR*>

« OK! DHCP<*CR*>

*Example 16. **GIP** command and answer (DHCP active)*

» **GIP**<*CR*>

« OK! AUTOIP<*CR*>

*Example 17. **GIP** command and answer (AutoIP active)*

» **GIP**<*CR*>

« OK! 192.168.2.239 255.255.255.0

*Example 18. **GIP** answer (static IP-address, no gateway configured)*

Instruction

GIP <*CR*>

Return Values in Case of Success

OK! {IP} {NetMask} [{Gateway}]
: Returned for static IP addressing. All IPs are <*SPACE*> separated and the trailing gateway address is optional (may be ommitted if it equals `0.0.0.0`).

"OK! DHCP <*CR*>" or "OK! AUTOIP <*CR*>"

Return Values in Case of Failure

"ERR <*CR*>" or "UPA <*CR*>"

1.14. Configure UART Line Settings (SUART)

SUART allows to configure various parameters of the specified UART interface.

Note

There also is a **UART** alias for TUC 1 backwards compatibility. The only difference to the TUC 1 version is that HARDWARE is not yet supported on the TUC 2.

Instruction

SUART <*SPACE*> {*UART*} <*SPACE*> {*Baudrate*} <*SPACE*> {*Stopbits*} <*SPACE*> {*Wordsize*} <*SPACE*> { NONE } <*SPACE*> { NONE | ODD | EVEN | MARK | SPACE } <*CR*>

Parameters

Name	Type	Description
UART	Decimal Integer ($0 \leq x \leq 1$)	Number of the UART to configure
Baudrate	Decimal Integer ($300 \leq x \leq 1024000$)	Desired baudrate of the interface
Stopbits	Decimal Integer ($1 \leq x \leq 2$)	Number of stopbits per word
Wordsize	Decimal Integer ($5 \leq x \leq 8$)	Number of bits per word
Flow Control	Enumeration (`NONE`)	
Parity	Enumeration (`NONE`, `ODD`, `EVEN`, `MARK` or `SPACE`)	

Examples

```
SUART 0 115200 1 8 NONE NONE<CR>
```

Example 19. Set UART 0 to 115200 baud 8N1

Return Values in Case of Success

`OK! <CR>`

Return Values in Case of Failure

`UPA <CR>`
May also be returned in case of invalid UART ids.

`EDX <CR>`
Baudrate, number of stopbits or wordsize are not valid integers

`NOR <CR>`
Baudrate, number of stopbits or wordsize are not in the supported range.

`NOS <CR>`
Flow Control is not yet configurable.

"`ERR <CR>`" or "`UPA <CR>`"

1.15. Get UART Line Settings (GUART)

Returns the current settings of the specified UART interface.

» **GUART** 0<*CR*>

« OK! 115200 1 8 NONE NONE<*CR*>

*Example 20. **GUART** command and answer*

Instruction

GUART <*SPACE*> {*UART*} <*CR*>

Parameters

Name	Type	Description
UART	Decimal Integer (0 ≤ x ≤ 1)	Number of the UART to query

Return Values in Case of Success

OK! {Baudrate} {Stopbits} {Wordsize} {Flow Control} {Parity}
Returns the UART line settings. `Flow Control` is currently always `NONE`. See Section 1.14, "Configure UART Line Settings (**SUART**)" for details.

Return Values in Case of Failure

"ERR <*CR*>" or "UPA <*CR*>"

1.16. Set UART mode (SMODE)

The **SMODE** command configures the kind of IP connection the specified UART can be accessed with. Each UART can be either driven in TCP client mode (TUC 2 connects to a remote server to transfer data) or in TCP server mode (TUC 2 waits for incoming connections).

Note

There is an alias **MODE** for TUC 1 compatibility. This command is largely compatible with the TUC 1, with the following exceptions:

- It can be used to turn off an UART (`SMODE OFF`).
- `TELNET` is unsupported by the TUC 2.
- The deprecated "idle timeout" setting, still present on the TUC 1 has finally been removed.
- UDP client mode (`NAME UDP CLIENT`) is still unimplemented on the TUC 2.

1.16.1. Disable UART (OFF)

Disable UART. Disabling UARTs improves latencies of the data transmission on the remaining UARTs.

Instruction

SMODE *<SPACE>* {*UART*} *<SPACE>* OFF *<CR>*

Parameters

Name	Type	Description
UART	Decimal Integer (0 ≤ x ≤ 1)	Number of the UART to disable.

Examples

```
SMODE 1 OFF<CR>
```

Example 21. Disable UART 1

```
MODE 1 OFF<CR>
```

Example 22. Disable UART 1 (legacy)

1.16.2. Configure server mode (SERVER)

Configure UART as a TCP server.

Instruction

SMODE *<SPACE>* {*UART*} *<SPACE>* { RAW } *<SPACE>* SERVER *<SPACE>* {*Local Port*} *<CR>*

Parameters

Name	Type	Description
UART	Decimal Integer (0 ≤ x ≤ 1)	Number of the UART to configure as server.
Protocol	Enumeration (RAW)	Protocol to use for incoming connections.
Local Port	Decimal Integer (1024 ≤ x ≤ 65535)	Local TCP port to listen for incoming connections.

Examples

```
SMODE 0 RAW SERVER 10001<CR>
```

Example 23. Configure UART 0 to accept connections on port 10001

1.16.3. Configure client mode (CLIENT)

Configure UART as a TCP client, ie. the TUC will attempt to establish an outgoing connection.

Instruction

SMODE *<SPACE>* {*UART*} *<SPACE>* { RAW } *<SPACE>* CLIENT *<SPACE>* {*…IP, Remote Port, Local Port…*} *<CR>*

Parameters

Name	Type	Description
UART	Decimal Integer ($0 \leq x \leq 1$)	Number of the UART to configure as client.
Protocol	Enumeration (RAW)	Protocol to use for outgoing connections.
IP, Remote Port, Local Port	Any String	Three arguments, separated by *<SPACE>* must be supplied: 1. IP address of the remote host, that the TUC will attempt to connect to. 2. Port on the remote host, where the TUC will connect to (between 1 and 65535). 3. Local port of outgoing connections (between 1024 and 65535).

Examples

```
SMODE 0 RAW CLIENT 192.168.2.42 80 10001<CR>
```

Example 24. Configure UART 0 to connect to 192.168.2.42:80

Return Values in Case of Success

`OK! <CR>`

Return Values in Case of Failure

`UPA <CR>`
May also be returned in case of invalid UART Ids.

`EDX <CR>`
One or more ports are not integers

`NOR <CR>`
One or more ports are out of the allowed range

"`NOS <CR>`", "`ERR <CR>`" or "`UPA <CR>`"

1.17. Get UART mode (GMODE)

GMODE returns the current mode by which the specified UART can be accessed.

» `GMODE 0<CR>`

« `OK! RAW SERVER 1337<CR>`

*Example 25. **GMODE** command and answer (server mode, listening on port 1337)*

» `GMODE 0<CR>`

« `OK! RAW CLIENT 192.168.2.101 44344 1337<CR>`

*Example 26. **GMODE** command and answer (client mode, connected to `192.168.2.101:44344` using local port `1337`)*

Instruction

GMODE *<SPACE>* {*UART*} *<CR>*

Parameters

Name	Type
UART	Decimal Integer (0 ≤ x ≤ 1)

Return Values in Case of Success

`OK! OFF` *<CR>*
This UART is disabled.

`OK! RAW SERVER {Local Port}`
This UART is a TCP server, listening on `Local Port`.

`OK! RAW CLIENT {Remote Address} {Remote Port} {Local Port}`
This UART is a TCP client, connecting to `Remote Address` at `Remote Port`. `Local Port` will be used as the source port for outgoing connections.

Return Values in Case of Failure

"ERR *<CR>*" or "UPA *<CR>*"

1.18. Set Client Reconnect Interval (SCLIENTRIV)

The **SCLIENTRIV** command allows to specifiy the interval between two consecutive tries when reconnecting to the remote host in client mode.

When a TCP connection, that is responsible for handling UART transfers, is unexpectedly closed, the TUC 2 will automatically try to reconnect to the remote in the specified interval.

Instruction

SCLIENTRIV *<SPACE>* {*UART*} *<SPACE>* {*Interval*} *<CR>*

Parameters

Name	Type	Description
UART	Decimal Integer ($0 \leq x \leq 1$)	Number of the UART whose reconnect interval should be set.
Interval	Decimal Integer ($x \geq 0$)	Interval in milliseconds

Examples

```
SCLIENTRIV 0 1000<CR>
```

Example 27. Set reconnect interval to 1 second

Return Values in Case of Success

`OK! <CR>`

Return Values in Case of Failure

`UPA <CR>`
May also be returned in case of invalid UART ids.

`EDX <CR>`
`Interval` value is not a valid interger.

"`ERR <CR>`" or "`UPA <CR>`"

1.19. Get Client Reconnect Interval (GCLIENTRIV)

Returns the client reconnect interval set by **SCLIENTRIV**.

» `GCLIENTRIV 0<CR>`

« `OK! 1000<CR>`

*Example 28. **GCLIENTRIV** command and answer*

Instruction

GCLIENTRIV *<SPACE>* {*UART*} *<CR>*

Parameters

Name	Type	Description
UART	Decimal Integer ($0 \leq x \leq 1$)	Number of the UART, for which to retrieve the client reconnect interval.

Return Values in Case of Success

`OK! {Interval} <CR>`
`Interval` is the client reconnect interval in milliseconds.

Return Values in Case of Failure

"`ERR <CR>`" or "`UPA <CR>`"

1.20. Configure TCP Keep Alive Probes (SKEEPALIVE)

The **SKEEPALIVE** command allows to change the TCP keep alive settings for the specified UART. TCP keep alive probes allow the TUC to detect "dead" connections without relying on payload traffic, which is important to free up resources and allow reconnects. They are active for both outgoing (client) and incoming (server) connections (see Section 1.16, "Set UART mode (**SMODE**)").

Note

The **KEEPALIVE** alias is supported for TUC 1 backwards compatibility. The only difference to **KEEPALIVE** on the TUC 1 is that `OFF` can be passed to disable TCP keep alive probes easily.

1.20.1. Disable TCP Keep Alives (OFF)

Disable TCP Keep Alive probes on the specified UART. This is equivalent to specifying `Idle Timeout` as 0.

Warning

Without TCP Keep Alive probes, dead connections cannot be detected automatically which may result in the inability to accept any new incoming connections. It should *never* be necessary to disable TCP keep alives.

Instruction

SKEEPALIVE *<SPACE>* {*UART*} *<SPACE>* `OFF` *<CR>*

Parameters

Name	Type	Description
UART	Decimal Integer ($0 \leq x \leq 1$)	Number of the UART on which Keep Alives should be disabled.

Examples

```
SKEEPALIVE 0 OFF<CR>
```

Example 29. Disable TCP Keep Alives for UART 0

```
KEEPALIVE 0 OFF<CR>
```

Example 30. Disable TCP Keep Alives for UART 0 (legacy alias)

1.20.2. Configure TCP Keep Alives

Configure TCP Keep Alive timeouts. The total time required to detect a "dead" connection is calculated by: `Idle Timeout` + `Counter` × `Interval`.

Instruction

SKEEPALIVE *<SPACE>* {*UART*} *<SPACE>* {*Idle Timeout*} *<SPACE>* {*Counter*} *<SPACE>* {*Interval*} *<CR>*

Parameters

Name	Type	Description
UART	Decimal Integer ($0 \leq x \leq 1$)	Number of the UART on which to configure Keep Alives.
Idle Timeout	Decimal Integer ($x \geq 0$)	Time in milliseconds of connection idleness before the first TCP Keep Alive probe is sent. Specifying 0 is equivalent to using the `OFF` parameter.
Counter	Decimal Integer ($x \geq 0$)	Number of TCP Keep Alive probes sent after the initial ones that must be unanswered before declaring a connection as "dead".
Interval	Decimal Integer ($x \geq 0$)	Interval in milliseconds between TCP Keep Alive Probes.

Examples

```
SKEEPALIVE 0 5000 5 1000<CR>
```

Example 31. Configure UART 0 to detect dead connections after 10 seconds

Return Values in Case of Success

OK! *<CR>*

Return Values in Case of Failure

UPA *<CR>*
May also be returned in case of invalid UART ids.

EDX *<CR>*
One of the integers may not be in the correct format.

"ERR *<CR>*" or "UPA *<CR>*"

1.21. Get TCP Keep Alive Settings (GKEEPALIVE)

Returns the TCP keep-alive settings of a specified UART, set by **SKEEPALIVE**.

» `GKEEPALIVE 0<CR>`

« `OK! 5000 5 1000<CR>`

*Example 32. **GKEEPALIVE** command and answer*

» `GKEEPALIVE 0<CR>`

« `OK! OFF<CR>`

*Example 33. **GKEEPALIVE** command and answer (TCP keep-alives disabled)*

Instruction

GKEEPALIVE *<SPACE>* {*UART*} *<CR>*

Parameters

Name	Type
UART	Decimal Integer (0 ≤ x ≤ 1)

Return Values in Case of Success

OK! {Idle Timeout} {Counter} {Interval}
The `Idle Timeout` and `Interval` are returned as integers representing milliseconds. `Counter` is also an unsigned integer. For details, see Section 1.20.2, "Configure TCP Keep Alives".

OK! OFF *<CR>*
Keep Alive Probes are disabled.

Return Values in Case of Failure

"ERR *<CR>*" or "UPA *<CR>*"

1.22. Set Flush Byte (SFLBYTE)

The **SFLBYTE** command allows to specify a special byte that is used as flush-indicator for the specified UART.

Normally data, that is received by the UART, is bufferd internally until some specific conditions regarding packet size and traffic flow are met before it is sent via TCP. This is done to increase the available network bandwidth at cost of latency. For systems where latency is important **SFLBYTE** can be used to specify a magic-byte that causes the TUC 2 to flush its internal buffers

and therefore bypass these mechanisms, while still saving some bandwidth. Whenever this byte is received, all data buffered by the specified UART will be sent as soon as possible.

Note

For backwards-compatibility with the TUC 1, there is a **FLBYTE** alias.

Instruction

SFLBYTE *<SPACE>* {*UART*} [*<SPACE>* {*Code*}] *<CR>*

Parameters

Name	Type	Description
UART	Decimal Integer (0 ≤ x ≤ 1)	Number of the UART, whose flush byte should be configured.
Code	Optional Decimal Integer (0 ≤ x ≤ 255)	The desired flush-byte as ASCII encoded decimal. If omitted, data is not flushed based on received bytes.

Examples

```
SFLBYTE 0 127<CR>
```

Example 34. Set flush-byte of UART 0

```
SFLBYTE 0<CR>
```

Example 35. Deactivate flush-byte feature

Return Values in Case of Success

OK! *<CR>*

Return Values in Case of Failure

UPA *<CR>*
: May also be returned in case of invalid UART ids.

EDX *<CR>*
: Flush-byte is not a valid integer.

NOR *<CR>*
: Flush-byte is out of range.

"ERR *<CR>*" or "UPA *<CR>*"

1.23. Get Flush-Byte (GFLBYTE)

Returns the flush-byte configured by **SFLBYTE**.

» `GFLBYTE 0<CR>`

« `OK! 127<CR>`

*Example 36. **GFLBYTE** command and answer (get flush-byte of UART 0)*

Instruction

GFLBYTE *<SPACE> {UART} <CR>*

Parameters

Name	Type	Description
UART	Decimal Integer (0 ≤ x ≤ 1)	Number of the UART to query

Return Values in Case of Success

OK! {Code}
Value of the flush-byte in decimal representation.

Return Values in Case of Failure

"ERR *<CR>*" or "UPA *<CR>*"

1.24. Set Send Timeout (SSENDTT)

Sets the minimum time between receiving a byte on the specified UART and sending the buffered data via the associated TCP connection. If the timeout value is set to any value greater than 0, every byte received will restart a timeout of the configured interval, delaying the output of all buffered data bytes to the TCP connection.

The buffered data will finally be output if:

1. The timeout expired without a byte being received by the UART
2. When a flush-byte is received, in case this feature is activated (refer to **SFLBYTE**)
3. The internal buffer reaches the maximum TCP payload size (normally 1460 bytes)

Setting the timeout to 0 deactivates the delay and received bytes are sent as soon as possible. Nevertheless a TCP packet may still contain multiple bytes.

Note

For compatibility with the TUC 1, there is a **SENDTT** alias.

Instruction

SSENDTT *<SPACE> {UART} <SPACE> {Value} <CR>*

Parameters

Name	Type	Description
UART	Decimal Integer (0 ≤ x ≤ 1)	Number of the UART to configure
Value	Decimal Integer (x ≥ 0)	Timeout value in milliseconds

Examples

```
SSENDTT 0 50<CR>
```

Example 37. Set send timeout of UART 0 to 50 milliseconds

Return Values in Case of Success

`OK! <CR>`

Return Values in Case of Failure

`UPA <CR>`
May also be returned in case of invalid UART ids.

`EDX <CR>`
Timeout value is not a valid integer.

"`ERR <CR>`" or "`UPA <CR>`"

1.25. Get Send Timeout (GSENDTT)

Returns the send timeout configured by the **SSENDTT** command.

» `GSENDTT 0<CR>`

« `OK! 50<CR>`

*Example 38. **GSENDTT** command and answer*

Instruction

GSENDTT *<SPACE> {UART} <CR>*

Parameters

Name	Type	Description
UART	Decimal Integer (0 ≤ x ≤ 1)	Number of the UART to query

Return Values in Case of Success

OK! {Send Timeout}
The configured send timeout in milliseconds.

Return Values in Case of Failure

"ERR *<CR>*" or "UPA *<CR>*"

1.26. Configure SNMP (SNMP)

Sets/Gets various SNMP settings.

1.26.1. Set SNMP standards (SCONF)

Sets which protocol standards are supported/enabled.

Instruction

SNMP *<SPACE>* SCONF [*<SPACE>* V1] [*<SPACE>* V2C] [*<SPACE>* V3] *<CR>*

Parameters

Name	Type	Description
V1	Flag	Enable protocol version V1
V2C	Flag	Enable protocol version V2C
V3	Flag	Enable protocol version V3C

Examples

```
SNMP SCONF V1 V2C<CR>
```

Example 39. Enable SNMP protocol versions V1 and V2C

1.26.2. Get SNMP standards (GCONF)

Returns currently enabled SNMP protocol standards.

» `SNMP GCONF<CR>`

« `OK! V1 V2C<CR>`

*Example 40. **GCONF** command and answer*

Instruction

SNMP *<SPACE>* GCONF *<CR>*

1.26.3. Set SNMP communities (SCOMM)

Sets the SNMP read-only and read-write communities. They are only relevant when SNMP V1 or V2C has been enabled.

Instruction

SNMP *<SPACE>* SCOMM *<SPACE>* {*…Read-Only, Read-Write…*} *<CR>*

Parameters

Name	Type	Description
Read-Only, Read-Write	Any String	Two alphanumerical strings containing the read-only community and the read-write one. Both strings are seperated by a *<SPACE>*. The Community strings themselves may not contain any whitespace or *<CR>* and are at most 32 characters long.

Examples

```
SNMP SCOMM public private<CR>
```

Example 41. Set SNMP communities

1.26.4. Get SNMP communities (GCOMM)

Returns the configured SNMP read-only and read-write communities.

» `SNMP GCOMM<CR>`

« `OK! public private<CR>`

*Example 42. **GCOMM** command and answer*

Instruction

SNMP *<SPACE>* GCOMM *<CR>*

1.26.5. Set SNMPv3 credentials (SUSR)

Sets SNMP version 3 credentials.

Instruction

SNMP *<SPACE>* SUSR *<SPACE>* {*…Credentials…*} *<CR>*

Parameters

Name	Type	Description
Credentials	Any String	The credentials consist of the following fields: 1. A username (without spaces), no more than 32 character wide.

Name	Type	Description
		2. Authorization algorithm (`MD5` or `SHA`) 3. Authorization password (at least 8 characters, without spaces) 4. Privacy algorithm (`DES` or `AES`) 5. Pivacy password (at least 8 characters, without spaces)

Examples

```
SNMP SUSR root MD5 maplesyrup AES maplesyrup<CR>
```

Example 43. Set SNMPv3 credentials

1.26.6. Get SNMPv3 credentials (GUSR)

Returns the configured SNMPv3 username and algorithms.

» `SNMP GUSR<CR>`

« `OK! root MD5 AES<CR>`

*Example 44. **GUSR** command and answer*

Instruction

SNMP *<SPACE>* GUSR *<CR>*

1.26.7. Set/Disable Trap Destination (STRP)

Sets or disables the trap destination by IPv4 address.

Instruction

SNMP *<SPACE>* STRP *<SPACE>* { OFF | IP4 } *<SPACE>* {*…Address…*} *<CR>*

Parameters

Name	Type	Description
Type	Enumeration (`OFF` or `IP4`)	
Address	Any String	The IP-address to set as trap destination. Depending on whether `IP4` or `IP6` was specified before, this string either contains an IPv4 or an IPv6 address. Omit if `OFF` was selected.

Examples

```
SNMP STRP IP4 192.168.2.3<CR>
```

Example 45. Set SNMP trap destination to 192.168.2.3

```
SNMP STRP OFF<CR>
```

Example 46. Disable SNMP trap delivery

1.26.8. Get SNMPv3 Trap Destination (GTRP)

Returns the current state/the configured address of trap destination.

» `SNMP GTRP<CR>`

« `OK! IP4 192.168.2.3<CR>`

*Example 47. **GTRP** command and answer*

Instruction

SNMP *<CR>*

Return Values in Case of Success

OK! *<CR>*
Returned by the setters (SCONF, SCOMM, SUSR and STRP) in case of success.

Return Values in Case of Failure

"ERR *<CR>*" or "UPA *<CR>*"

1.27. Set Discovery Protocols (SDISCOVERY)

The TUC 2 features proprietary protocols to simplify device discovery in networks with dynamically assigned IP-addresses (DHCP or AutoIP). This protocol is used by all metraTec tools and enables the user to perform software updates or configure devices without knowing their IP-address. The **SDISCOVERY** command allows to select (or disable) the protocol used for this feature.

Instruction

SDISCOVERY [*<SPACE>* LEGACY] *<CR>*

Parameters

Name	Type	Description
LEGACY	Flag	Enables the "legacy" (TUC 1 compatible) discovery protocol. It should not be enabled in security-relevant use cases and does not work for IPv6.

Examples

```
SDISCOVERY LEGACY<CR>
```

Example 48. Enable legacy discoveery protocol

```
SDISCOVERY<CR>
```

Example 49. Disable all discovery protocols

Return Values in Case of Success

OK! *<CR>*

Return Values in Case of Failure

"ERR *<CR>*" or "UPA *<CR>*"

1.28. Get Discovery Protocols (GDISCOVERY)

Returns the activated protocols for device discovery set by **SDISCOVERY**.

» **GDISCOVERY**<*CR*>

« OK! LEGACY<*CR*>

*Example 50. **GDISCOVERY** command and answer (legacy discovery protocol enabled)*

Instruction

GDISCOVERY *<CR>*

Return Values in Case of Success

OK! [{Protocols}] *<CR>*
: OK! if no protocols are enabled, followed by a list of protocols otherwise. Currently only the LEGACY protocol is supported.

Return Values in Case of Failure

"ERR *<CR>*" or "UPA *<CR>*"

1.29. Apply Settings (APPLY)

This command applies all staged setting changes, thus making them persistent. The device will automatically reboot after the changes have been applied.

» **APPLY**<*CR*>

« OK!<CR>
REBOOT<CR>

*Example 51. **APPLY** command and answer*

Instruction

APPLY *<CR>*

Return Values in Case of Success

OK!<CR>REBOOT<CR>
Unlike most other commands, this returns two *<CR>*-separated response lines.

Return Values in Case of Failure

"ERR *<CR>*" or "UPA *<CR>*"

1.30. Discard Settings (DISCARD)

This command discards all staged setting changes. No changes are made to the internal parameter storage.

Instruction

DISCARD *<CR>*

Return Values in Case of Success

OK! *<CR>*

Return Values in Case of Failure

"ERR *<CR>*" or "UPA *<CR>*"

1.31. Restore Factory Settings (FACTORY)

The **FACTORY** command resets all settings to their factory defaults. Refer to Appendix A, *Factory Defaults* for information about the default settings.

» **FACTORY***<CR>*

« OK!<CR>
REBOOT<CR>

*Example 52. **FACTORY** command and answer*

Instruction

FACTORY *<CR>*

Return Values in Case of Success

OK!<CR>REBOOT<CR>
Unlike most other commands, this returns two *<CR>*-separated response lines.

Return Values in Case of Failure

"ERR *<CR>*" or "UPA *<CR>*"

1.32. Reset Device (RESET)

RESET performs a system reset/reboot. When executing this command all network connections are dropped, the WebUI becomes unavailable and data transfered via UART may be lost. The device will behave like after (re-)powering and is set to its default operational state.

A new connections can only be established after the restart process has finished. This may take several hundred milliseconds.

» **RESET***<CR>*

« OK!*<CR>*
REBOOT*<CR>*

*Example 53. **RESET** command and answer*

Instruction

RESET *<CR>*

Return Values in Case of Success

OK!<CR>REBOOT<CR>
Unlike most other commands, this returns two *<CR>*-separated response lines.

Return Values in Case of Failure

"ERR *<CR>*" or "UPA *<CR>*"

1.33. Close Connection (CLOSE)

CLOSE lets the TUC 2 close the current Config Shell connection, which has the same effect as if the client closed its TCP connection. Naturally, no more commands can be sent after **CLOSE** has been invoked until a new connection to the Config Shell is opened.

Note

Unlike all other commands, this command has *no* return value. This is for compatibility with the TUC 1.

It should not be necessary to use this command at all since a client can can always terminate its TCP connection without informing the TUC first.

Instruction

CLOSE *<CR>*

Return Values in Case of Success

Return Values in Case of Failure

"ERR *<CR>*" or "UPA *<CR>*"

Chapter 2. Error Codes

Error Code	Name	Description
ALREADYSET	Already Set	A one-time setting (eg. serial number or MAC address) has already been programmed.
BOF	Buffer Overflow	An internal buffer overflowed. This should not be possible to happen, as long as sending redundant whitespace characters to the device is avoided.
EDX	Error Decimal Expected	Parameter string cannot be interpreted as a valid decimal value. Common error source: • Other character than '0' to '9' sent
EHX	Error Hex Expected	Parameter string cannot be interpreted as a valid hexadecimal number or string.
ERR	Miscellaneous Error	Otherwise unhandled or unexpected errors might sometimes be reported with the **ERR** error code (as a catch-all error code).
NOR	Number Out of Range	One of the integer parameters is out of the supported range.
NOS	Not Supported	Command or parameter not supported.
NOTSET	Not Set	A queried setting (eg. serial number) has not yet been programmed.
UCO	Unknown Command	An invalid command has been sent to the TUC. Common error sources: • Typo in command string • Wrong firmware version
UPA	Unknown Parameter	An invalid parameter has been passed to a function. Common error source: • Typo in command string • Given parameter is out of range • Parameter missing

Appendix A. Factory Defaults

Property	Default Value
Module Name	metraTec TUC 2 Module
Address Type	Static IPv4
IP Address	192.168.2.239
Subnet Mask	255.255.255.0
Default Gateway	192.168.2.1
DNS	8.8.8.8, 8.8.4.4
Enabled UART to TCP Ports	UART 0, UART 1
Baudrate (all UARTs)	115200 baud
Data Size (all UARTs)	8 bits
Parity (all UARTs)	None
Stop Bits (all UARTs)	1
Flow Control (all UARTs)	None
Connection Mode (all UARTs)	Server
Local Port (UART 0)	10001
Local Port (UART 1)	10002
Flush Byte (all UARTs)	Disabled
Send Timeout (all UARTs)	0 milliseconds
Client Reconnect Interval (all UARTs)	3000 milliseconds
SNMP Protocols	V3
SNMPv3 Username	`admin`
SNMP Authentication and Privacy Passwords	`maplesyrup`
SNMP Authentication Algorithm	MD5
SNMP Privacy Algorithm	AES
TCP Keepalive Idle Timeout (all UARTs)	5000 milliseconds
TCP Keepalive Counter (all UARTs)	5
TCP Keepalive Interval (all UARTs)	1000 milliseconds
Authorization Username	`admin`
Authorization Password	`tucadmin`
Control Shell (Port 40000)	Enabled
Discovery Protocols	Legacy

Version Control

Version	Change	By	Date
2.0-RC1	created	TP, RH	01.04.2019

metraTec GmbH

Niels-Bohr-Str. 5
39106 Magdeburg
Germany

Tel.: +49 (0)391 251906-00
Fax: +49 (0)391 251906-01

Email: <support@metratec.com>
Web: http://www.metratec.com