SRCP - Simple Railroad Command Protocol 0.7.3

Torsten Vogt, Martin Ostermann, Kurt Haders, Olaf Schlachter, Matthias Trute (Maintainer), Tobias Schlottke, Edbert van Eimeren, Stefan Bormann, Michael Reukauff, Martin Wolf 2000, 2001

SRCP describes a command set for client-server-communication between a server prozess controlling a digital model railway and its clients. A server prozess is either a software signal-generator or a dreiver for a hardware interface. clients are typecally controlling programs.

Contents

1 Introduction

For the english translation only: SRCP is developed in discussion on the newsgroup de.rec.modelle.bahn. So until a first final release the german version of this document availible at http://www.der-moba.de/Digital is the binding version. Translation was done by Martin Wolf.

A SRCP server reads on the one hand information of a model railway layout an make it availible for the SRCP clients in a TCP/IP network. On the other hand it executes the client's commands on the layout. A SRCP server has neither information of the layout (plan of track, wiring etc.) nor it cares for a status (setting of points etc.).

The main purpose is unified calling of different digital controlling systems. A client has not to know anything about the details of these systems.

The SRCP command set consists of three types of commands:

- commands controlling the server
- commands controlling model railway decoders
- commands for processing the feedback of the layout

A server can implement own information sources like a clock modul, that sends a model time to all clients.

1.1 Conventions

The client transmits its commands to the server via TCP/IP. All commands finish with the character '\n' (line feed (LF), ASCII #10). A previous '\r' (carriage return (CR), ASCII #13) is allowed. Every command consists of words seperated by a whitespace (space, tabulator).

Transmission of information from server th client is subject to the same conventions as reciving commands. A server can send either '\r\n' or '\n' as end of command. A client must interpret both as correct end of command.

The words of a command can consist of the characters { ".", ";", "0".."9", "*", "-", "A".."Z", "a".."z" }.

A server interprets the commands case-sensitive, i.e. it differences between upper and lower case characters.

Numbers are always integer. The range of value is not generally limited.

The server ignores incomplete or wrong commands. After some commands the client expectes server's answer. This has to be generated sensiblly and sended in every case. Unknown commands are ignored without any answer.

1.2 Transmission Channels

A SRCP server offers three TCP/IP-Ports to the clients. Standard port for the command port is 12345. Feedback port and information port are always the two direct following ports. So the standard port numbers are

- **command port** 12345
- **feedback port** 12346
- **information port** 12347

A change of the port numbers is allowed, if the given order is preserved.

On the command port clients transmits commands to a server. The server executes the commands. After some commands the client expectes an answer on the command port.

Connects a client to the command port, the server must send an one-line information text to the client on this port. This text consists of elements devided by a semicolon ';'. The first word of such an element is the key, all following words until the next semicolon or linefeed are the value. A closing semicolon in front of the linefeed is allowed.

The information text have to contain following keyes:

- SRCP: the value is the SRCP version number implemented and supported. This must be one of the published values.

Example:

```
erddcd v0.9.511; SRCP 0.5.0
```

After sending the information text the server is waiting for client's commands. The client have to read the information text and can now send commands to the server.

The feedback port is used only in the direction from the server to the clients. Over this port clients are informed about changes on the layout.

Logs a client in the feedback port, the server sends all occupied feedback units and then every status change of every feedback unit to the client. This allows a event-controlled feedback handling of the clients.

Also the information port is used only in the direction from the server to the clients. It is a kind of broardcast-channel continously feeded by the server with status changes by other clients or possible existent information sources of the layout (not feedback units!).

2 Commands Controlling The Server

Communication Ports

- **client -> server** command port
- **server -> client** not used

Kommandos

- **SHUTDOWN** The server should quit itself.
- **LOGOUT** Signals to the server a client logs out.
- **RESET** The server should reinitialize itself.

Following commands must not be used by the clients, but have to be implemented and supported by the server.

- STARTVOLTAGE identical to SET POWER ON
- STOPVOLTAGE identical to SET POWER OFF

3 commands controlling model railway decoders

3.1 Commands And Devices

SRCP defines 8 general commands handled over the command port

SET
: Set a value from the client via the server on the devive.

GET
: Get the current status of the device.

WAIT
: Waits until the device has reached a desired status.

INIT
: Used if a device needs a dedicated initialization.

TERM
: Removes the setting done by INIT.

READ
: Reads a value, enhancemant to WRITE

WRITE
: Sets a value from the client and returns an answer.

VERIFY
: Veryfies a value.

The devices are diveded into four device groups:

- locomotive decoders
- accessory (switching) decoders
- feedback units
- other equipment

The first parameter of a caoomand specefies the device group the command is addressed to.

The commands WRITE, VERIFY und READ are used together with programming interfaces to program decoders.

GL

loco and function decoder (generic loco)

GA

accessory (switching) decoders (generic accessory)

FB

feedback units (feedback)

TIME

time standard

POWER

power unit of the model railway

Using commands with device groups: round brackets mean optional elements not every server have to support. But pay attention, that for example every server have to accept WRITE. Nevertheless it have not to execute the command if the requirements are not met. TIME can miss totally.

	SET	GET	WAIT	INIT	TERM	WRITE	READ	VERIFY
GL	x	x	-	-	-	x	x	x
GA	x	x	-	-	-	x	x	x
FB	-	x	x	x	(x)	-	-	-
TIME	(x)	(x)	(x)	(x)	(x)	-	-	-
POWER	x	x	-	-	-	-	-	-

Pay attention on the difference between SET/GET and WRITE/READ. For some device groups both command pairs are specefied, but the parameters and the purpose are compleatly different.

After the commands GET, WAIT, WRITE and READ the client expectes an answer from the server. In the case, the server is not able to give an answer to the request, it have to send a one-line string of following format:

```
INFO <error code>
```

"error code" is a negative integer with following spezification:

INFO -1

==> command not supported

INFO -2

==> no data availible

INFO -3

==> timeout (see WAIT)

4 Commands for the Device Groups

For every device group the effect of the commandes is explained.

4.1 Generic Loco GL

For controlling of locomotive decoders and other mobile devices with all their additional functions.

4.1.1 SET GL

```
SET GL <protocol> <addr> <direction> <V> <V_max> <func> <nro_f> <f1> .. <fn>
```

communication ports

- **client** -> **server** command port
- **server** -> **client** not used

meaning of the parameters

protocol

L, M, N, S, PS

- **L** reserved for LocoNet®
- **M1** Märklin old (rel. dc, 80 Adr., 1 funct., 14 ss)
- **M2** Märklin new (abs. dc, 80 Adr., 5 funct., 14 ss)
- **M3** M2 enhanced (abs. dc, 256 Adr., 5 funct., 28 ss)
- **M4** M2 enhanced (abs. dc, 256 Adr., 5 funct., 14 FS)
- **M5** M2 enhanced from Märklin (abs. dc, 80 Adr, 5 funct., 27 FS)
- **MF** old Marklin format for funktions decoder
- **NB** NMRA-DCC basic protocol (abs. dc, 7-bit-Adr., 14 FS)
- **N1** NMRA-DCC enhanced (abs. dc, 7-bit-addr., 5/9/13 funct., 28 FS)
- **N2** NMRA-DCC enhanced (abs. dc, 7-bit-addr., 5/9/13 funct., 128 FS)
- **N3** NMRA-DCC enhanced (abs. dc, 14-bit-addr.,5/9/13 funct., 28 FS)

- **N4** NMRA-DCC enhanced (abs. dc, 14-bit-addr., 5/9/13 funct., 128 FS)
- **S** reserved for Selektrix
- **PS** protocol by server, the server selects the protocol

(ss = speed steps; dc = direction coding)

addr

integer >= 0

direction

0 (= reverse), 1 (= forward), 2 (= emergency stop)

V

0 .. V_max ((virtual) speed)

V_max

0 .. <integer > 0> (maximum (virtual) speed) V_max=0 ==> virtual speed step = real speed step

func

0 (= off), 1 (= on)

nro_f

0 .. x (number of functions)

f1 .. fn

0 (= off), 1 (= on)

An example:

```
SET GL N2 1 1 50 250 1 4 0 1 0 0
```

The loco with an enhanced NMRA-DCC decoder with address 1 goes with 1/5th of maximum speed forward. Function and F2 are swiched on.

Calculating speed steps from virtual speed:

given:

- **DEC_fs** number of speed steps the decoder implements (known throuh protocol specification)
- **V** virtual speed (parameter)
- **V_max** maximum virtual speed (parameter)

unknown:

- **V_fs** real speed step equivalent to virtual speed

algorithm: V_fs = round((V * DEC_fs)/V_max)

Note for developers of SRCP servers: Pay attention, that the real speed step 0 (stop) is only send if the parameter V is 0 (zero). The function "round" has to be implemented intelligent enough.

V_fs has to be interpreted as a speed value only. Some decoder react e.g. at speed step 1 with a emergency stop, other with a direction change. For the support of such decoders the server has to take care of adjusting V_fs before sending the command to the decoder. For the client the speed steps have to be numbered continously from 0 to maximum.

If V_max is set to 0, the server is not allowed to do any calculation on speed steps, i.e. the speed step in parameter V has to be sent direct to the decoder. In that way, the clients have direct access to the decoder.

If a client sets the protocol to PS (protocol by server), the server has to decide which protocol to select for the transmitted address. All other protocols also are recommendations of the client to the server. It can always choose an other, more usable protocol.

Examples

```
(50*28)/250 = 5.7   ==> V_fs = 6
(4*28)/250  = 0.448 ==> V_fs = 1 (!!!)
(0*28)/250  = 0     ==> V_fs = 0
```

4.1.2 GET GL

```
GET GL <protocol> <addr>
```

communication ports:

- **client** -> **server** command port
- **server** -> **client** command port

meaning of the parameters: look at 4.1.1 (SET GL)

The server sends all availible information about the loco or function decoder with <protocol> und <addr> to the client. This information has to have following format:

```
INFO GL <protocol> <addr> <direction> <V> <V_max> <func> <nro_f> <f1> .. <fn>
```

vgl. 4.1.1 (SET GL)

If no information is availible or the server does not support the GET command, the server have to answer with "INFO <error code>" to the client's request.

4.2 Generic Accessoire GA

With generic accessoires can decoders for lighting, points and signals etc. be controlled. They have independent accessible ports, which can be set to different statuses. Status 0 is the basic one and the server can set a port to this after a given delay time.

4.2.1 SET GA

```
SET GA <protocol> <addr> <port> <action> <delay>
```

communication ports:

- **client** -> **server** command port
- **server** -> **client** not used

meaning of the parameters

protocol

- **M** Märklin/Motorola-format
- **N** NMRA-DCC-format
- **P** protocol by server, the server selects the protocol

addr

integer >= 0 (number of points/signal)

port

0, 1, 2,... (output port of the decoder)

action

0, 1, 2, ... (0 => deactivated, >0 => activated)

delay

value in milli seconds (1000th-seconds). After the given delay time the server will automatically deactivate an activated output port. If <delay> is set to "-1" the output port will not be automatically deactivated. If <action> = 0 (deactivation) delay is ignored, but it must be set (useful value: "-1").

A port is justed as active if its <action> status is not equal to 0 (zero).

Connecting the decoder output ports:

- **0** = points turning, signal "stop", ...
- **1** = points straight, signal "go", ...

Example:

```
SET GA M 23 1 1 20
```

This is activating GA number 23, port 1 for 20 ms. Attention: there may be minimal activating times for some systems. It is also possible that dependencies between the ports of an output or between two outputs are existing.

4.2.2 GET GA

```
GET GA <protocol> <addr> <port>
```

communication ports:

- **client** -> **server** command port
- **server** -> **client** command port

meaning of the parameters: look at 4.2.1 (SET GA)

The server sends all availible information about the actual state of the general accessory decoder spezified through <protocol> and <addr>. This information has the format:

```
INFO GA <protocol> <addr> <port> <state>
```

(vgl. 4.2.1 (SET GA), replace <state> with <action>)

If there is no information availible or the server does not support the GET command, it has to send the defined error code "INFO <error code>" to the client.

4.3 Time Standard TIME

The time standard is used for supporting all clients with an uniform model time. It's optional, i.e. a SRCP server must not support it.

4.3.1 INIT TIME

```
INIT TIME <JulDay> <Hour> <Minute> <Second> <fx> <fy>
```

Starts the time standard with the scaling factor FX/FY at the given time. Every full model minute the server sends an INFO-string on the INFO-Port to all clients logged in there.

The model time is calculated as follows:

```
(Delta) model time = (Delta) real time * FX / FY
```

Example:

- FX=10 FY=1 -> There will be 10 model minutes in one real minute, every six second one.
- FX=1 FY=10 -> One model minute lasts 10 real minutes.
- FX=1 FY=1 -> The model time is equivalent to the real time.

The day is increased every 24 model hours.

4.3.2 SET TIME

```
SET TIME <JulDay> <Hour> <Minute> <Second> <fx> <fy>
```

communication ports:

- **client -> server** command port
- **server -> client** not used

meaning of the parameters

The actual model time and scaling to real time is defined. For the first calling compareable to INIT TIME.

JulDay

Julian Daynumber (look at glossary)

Hour

0..23, consists of 60 minutes

Minute

0..59, consists of 60 seconds

Second

0..59

FX, FY

integers for time scaling

Example

```
SET TIME 1 23 55 0 1 1
```

(Model and real) time is set to five to midnight at day one with a scaling of 1. (look at 4.3.1 (INIT TIME))

4.3.3 GET TIME

```
GET TIME
```

communication ports:

- **client** -> **server** command port
- **server** -> **client** command port

Get the actual model time with scaling als INFO-string:

```
INFO TIME <JulDay> <Hour> <Minute> <Second <fx> <fy>
```

Is there no model time defined, the server reports this with "INFO -2" (no data) th the client. (look at 3.1 (Commands and Devices)).

4.3.4 WAIT TIME

```
WAIT TIME <JulDay> <Hour> <Minute> <Second>
```

communication ports:

- **client** -> **server** command port
- **server** -> **client** command port

Waits until the model time has reached the given time or more and sends an INFO-string with the actual model time to the server.

If the time standard is not initialized the server sends "INFO -1". Is the actual model time greater as the given time the requirement is matched without any more waiting time. Wrong time parameters are responded with "INFO -1".

4.3.5 TERM TIME

communication ports:

- **client** -> **server** command port
- **server** -> **client** command port

This stops the time standard. It is optional.

4.4 Power Unit POWER

4.4.1 SET POWER

```
SET POWER <state> <freetext>
```

communication ports:

- **client** -> **server** command port
- **server** -> **client** not used

meaning of the parameters

State

ON
swiches the power unit on

OFF
swiches the power unit off

Freetext

An optional text with a maximum of 100 characters added to the INFO-string for additional information. For the content there is no requirement.

4.4.2 GET POWER

```
GET POWER
```

communication ports:

- **client** -> **server** command port
- **server** -> **client** command port

Gets the actual state of the power unit. The server responds with

```
INFO POWER [ON|OFF] <freetext>
```

<Freetext> it is the value last set or an other value generated by the server. (cd. 4.4.1 (SET POWER))

4.5 Feedback devices FB

4.5.1 INIT FB

```
INIT FB <module_type>
```

communication ports:

- **client** -> **server** command port
- **server** -> **client** not used

The server initialises the feedback subsystem. This initialisation enables the feedback information and the feedback port. The client does not get any response. If the server executes this command itself (e.g. on program start), then this command will be ignored.

4.5.2 GET FB

```
GET FB <module_type> <portnr>
```

communication ports:

- **client** -> **server** command port
- **server** -> **client** command port

meaning of the parameters

module_type

- **S88** Marklin s88-bus on the parallel-port of the PC
- **I8255** i8255 board
- **M6051** Marklin s88-bus via Marklin-interface 6051
- **PS** protocol by server: the server decides what protocol to use

portnr

number of an input of a feedback modul or oder `"*"` for all possible inputs

On the command port the server sends the actual state of the specified (<module_type> and <portnr>) feedback input. This information has to use the following format:

```
INFO FB <module_type> <portnr> <state>
```

`state` is only allowed to be `"0"` or `"1"`.

If the wildcard `"*"` was used as `portnumber`, so the server sends as `portnr` the wildcard `"*"` (without the doublequotes) followed by the states of all connected ports. This states will not be divided trough whitespaces. Be aware that this string could get very long!

Example

```
INFO FB M6051 * 1100110010101111
```

If there is no information availible or the server does not support the GET command, it has to send "INFO <error code>" to the client (cd. 3.1 (Commands And Devices)).

4.5.3 WAIT FB

```
WAIT FB <module_type> <portnr> <value> <timeout>
```

communication ports:

- **client** -> **server** command port
- **server** -> **client** command port

Waits, until the specified port has the value `value` or a timeout is reached afer `timeout` seconds. If timeout is reached, "INFO -3" is sent to the client. Otherwise the same information as for GET FB is sent.

If there is no information availible or the server does not support the WAIT command, it has to send "INFO <error code>" to the client (cd. 3.1 (Commands And Devices)).

4.5.4 TERM FB

```
TERM FB <module_type>
```

communication ports:

- **client** -> **server** command port
- **server** -> **client** not used

The server deactivates the corresponding feedback subsystem. This is an optional command.

5 Commands to program decoders

Some loco decoders are programmable on a special programming track or "on-track" in normal operation. This decoders allow the changing of memory-cells (register, configuration variable CV, bit) to influence the behavior of the decoder. This decoders can be supported by a SRCP server. For that purpose the commands WRITE, VERIFY and READ are specified.

In INFO lines as divice group the code SM is used: Secvice Mode

5.1 Changing decoder settings

```
WRITE GL <protocol> <dest-type> <dest-addr> <value>
WRITE GA <protocol> <dest-type> <dest-addr> <value>
```

communication ports:

- client -> server: command port
- server -> client: command port

meaning of the parameters

protocol

NMRA, ...

dest-type

Type of the part of decoder memory that should be changed (destination type)

CV

NMRA-DCC configuration variable

CVBIT

one bit of a NMRA-DCC configuration variable, the content of dest-addr is then two words: the number of the CV and the bit number (0-7) in this CV

REG

a register of a NMRA-DCC-dekoder

dest-addr

address of the memory cell that should be changed. (if CVBIT is choosen as `dest-type`, the second word of addresse is the bit number (0-7).

value

new value of the memory cell

The SRCP server always sends information to the client in the following format, after it has received and executed a WRITE command.

```
INFO GL SM <value>
INFO GA SM <value>
```

<value> can take one of the following values:

- 0: WRITE failed.
- 1: WRITE succeeded
- 2: The server does not know the result of the WRITE command.

If WRITE or a special parameter of WRITE is not supported by the server, the information "INFO -1" is sent to the client on the command port.

5.2 Verify decoder settings

```
VERIFY GL <protocol> <dest-type> <dest-addr> <value>
VERIFY GA <protocol> <dest-type> <dest-addr> <value>
```

communication ports:

- client -> server: command port
- server -> client: command port

meaning of the parameters

protocol

NMRA, ...

dest-type

type of decoders memory cell to be verified. (destination type)

CV

NMRA-DCC configuration variable

REG

a register of a NMRA-DCC-dekoder

dest-addr

address of the memory cell to verify

value

value to verify in the memory cell

The SRCP server always sends information to the client in the following format, after it has received and executed a WRITE command.

```
INFO GL SM <value>
INFO GA SM <value>
```

<value> can take one of the following values:

- 0: The value in <value> is not the current value of the memory cell.
- 1: The value in <value> is the current value of the memory cell.
- 2: The server could not perform the VERIFY command.

If VERIFY or a special parameter of VERIFY is not supported by the server, the information "INFO -1" is sent to the client on the command port.

5.3 Reading decoder settings

Reserved for future enhancements.

6 serverinformationports

The serverinformationports are used for information of the clients about changes on the layout (feedback) and the executed commands of the SRCP server. The last one could only be handled as hints for the real state of the layout.

6.1 Feedback port

Data have to be sent with the following format:

```
INFO FB <module_type> <portnr> <state>
```

(cp. 4.5.2 (GET FB))

If the port is opend by a client, all ports with (state == 1) will be immedeately reported to the client. After that initial information, only changes are transmitted.

6.2 Information port

Data have to be sent with the following format dependent on the device group:

```
INFO GL <protocol> <addr> <direction> <V> <V_max> <func> <nro_f> <f1> .. <fn>
INFO GA <protocol> <addr> <port> <state>
INFO TIME <JulDay> <Hour> <Minute> <Second> <FX> <FY>
INFO POWER ON|OFF <Freetext>
RESET
SHUTDOWN
```

No other data is allowed to be sent on this port, especially no "INFO <error>".

If the port is opened by a client, for every device out of the group GL and GA the actual state is reported, also the actual model time (TIME) and the state of POWER.

For the device group GA the actual state is the state of the single ports in the chronological order, they have swiched to. For every GA device there is one INFO-line per port.

For the device group GL the actual state is the INFO-line corresponding to the last SET command, if the server can't use other informations from the layout.

Information is transmitted only for those devices, which were used or for which the server can reliable get information about their state (e.g. getting information from the devices if this is supported.), and not for the whole adress space.

After that initial information, only changes of the available devices are transmitted, when they have reached the device or the server knows about it.

7 Ideas and future enhancement

Some ideas were and are touched in the discussion, but they have not been integrated in this version of SRCP. To take care of them and save them for future enhancements they are listed in this section. They are explicit not part of the Simple Railroad Command Protocol (SRCP).

7.1 Central configuration

Kurt Haders suggested a central configuration file. The main purpose of that file is it, to shift more knowledge from the clients to the server. A special format of the configuration file has not yet been specified. The SRCP is prepared through the protocol "PS" (protocol by server) for realisation.

7.2 Expand commands on other device groups

Matthias Trute suggested allowing th command WAIT also for device groups GL and GA. Also the introduction of wildcards ("*") for WAIT is possible. But there may be inconsistency in this cases.

7.3 Giving a receipt for commands

Martin Ostermann suggests receipts for every command. This could be done through the feedback port.

7.4 Reading the server configuration

Edbert van Eimeren suggested that clients should be able to read the configuration of the servers (new command: CONFGET).

7.5 Modularisation

There are some ideas to call modules trough SRCP to reach some special functions.

7.6 Locking of single devices

Martin Wolf suggested an optional command for locking single devices. With this command a client can get a device (GA, GL, FB, POWER, ...) for exclusive use. If a client ends the session (logout) or the communication is interrupted, the devices will be unlocked. The realisation could be done via the modularisation of the protocol. So there is the need for new commands: LOCK, UNLOCK, BREAKLOCK, or a substitution of the existing SET LOCK 1, SET LOCK 0, WAIT LOCK, GET LOCK etc.

8 Glossary

This section is to explain some expressions not in common knowledge but essential for SRCP.

Julian Daynumber

Serial numbering of days instead of a struktured calender. A new day begins every 24 hours calculated form specified startday and starttime. There are some starting points in time commonly used: repeating New Year at 0h:0:0 (like in C runtime) or a single event like 1.1.4713 BC 12:00 GMT in astronomy.

There are algorithms to calculate Julian Daynumber from calendar information and vice versa, e. g. in Collected Algorithms from CACM #199

9 Legal Stuff