# 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

Das SRCP beschreibt einen Befehlssatz zur Client-Server-Kommunikation zwischen Serverprozessen zur Steuerung von digitalen Modelleisenbahnen und deren Clients. Serverprozesse sind entweder Software-Signalgeneratoren oder Treiber für HW-Interfaces. Clients sind typischerweise Steuerungsprogramme.

## Inhaltsverzeichnis

# 1 Einleitung

Ein SRCP Server dient dazu, Informationen von einer Modellbahnanlage einzulesen und entsprechenden Clients in einem IP Netzwerk verfügbar zu machen. Im Gegenzug führt er Befehle der Clients auf der Modellbahnanlage aus. Ein SRCP Server pflegt keinen Anlagenstatus (Weichenstellung etc.) und verfügt nicht über Informationen über die Anlage selbst (Gleispläne, Verdrahtung usw).

Hauptzweck ist es, verschiedene Digitalsysteme einheitlich ansprechen zu können. Ein Client muß sich nicht mehr mit den Details einer Digitalsteuerung auseinandersetzen.

Der SRCP-Befehlssatz besteht aus Kommandos, die direkt das Verhalten des Servers betreffen, und aus Kommandos, die für die Dekoder der Modellbahnanlage bestimmt sind. Weiterhin werden Kommandos, die die Verarbeitung von Rückmeldungen betreffen spezifiziert.

Der Server kann über eigene Informationgeneratoren wie ein Zeitgebermodul verfügen, das alle Clients mit einer einheitlichen Modellzeit versorgt.

## 1.1 Konventionen

Die Übertragung der Kommandos von den Clients zum Server erfolgt via TCP/IP. Alle Kommandos werden mit dem Zeichen '\n' (line feed (LF), #10) abgeschlossen. Ein vorangestelltes '\r' (carriage return (CR), #13) wird akzeptiert. Jedes Kommando besteht aus Worten, die durch Whitespace (Leerzeichen, Tabulatoren) getrennt sind.

Die Übertragung von Informationen vom Server zum Client unterliegt den gleichen Konventionen wie der Empfang von Befehlen. Ein Server kann beim Zeilenende sowohl '\r\n' als auch nur '\n' senden. Ein Client muß beides als korrektes Zeilenende werten.

Die Worte der Kommandos können aus der Menge der Zeichen { ".", ";", "0".."9", "*", "-", "A".."Z", "a".."z" } gebildet werden.

Der Server wertet die Kommandos case-sensitive aus, d.h. zwischen Groß- und Kleinbuchstaben wird unterschieden.

Zahlen sind Ganzzahlen. Eine Beschränkung des Wertebereiches besteht nicht generell.

Kommandos, die unvollständig oder offensichtlich falsch sind, werden vom Server ignoriert. Bei einigen Befehlen erwartet der Client jedoch eine Antwort vom Server. Diese muß in jedem Fall sinnvoll generiert und gesendet werden. Unbekannte Befehle werden ohne Rückmeldung ignoriert.

## 1.2 Übertragungskanäle

Ein SRCP-konformer Server stellt den Clients drei TCP-Ports zur Verfügung.Standardportnummer für den Kommandoport ist 12345. Der Rückmeldepollport und der Informationsport sind immer die beiden unmittelbar folgenden Portnummern. Standardmässig ergeben sich somit folgende Portnummern:

- **Kommandoport** 12345
- **Rückmeldepollport** 12346
- **Informationsport** 12347

Eine Verlagerung der Portnummer ist zulässig, solange die drei Ports in der angegebenen Weise aufeiander folgen.

Der Kommandoport dient den Clients dazu, dem Server Kommandos zu übermitteln. Der Server führt die Kommandos aus. Bei einigen Kommandos erwartet der Client eine Antwort auf dem Kommandoport.

Nimmt ein Client zum Kommandoport Kontakt auf, muß der Server zunächst einen einzeiligen Informationstext über diesen Port zum Client schicken. Dieser Text besteht aus durch ein Semikolon zu trennenden Angaben. Das erste Wort eines solchen Elements ist als Schlüssel, alle nachfolgenden Worte bis zum folgenden Semikolon bzw. dem Zeilenende sind der Wert. Ein abschließendes Semikolon vor dem Zeilenende ist zulässig.

Folgende Schlüssel sind zwingend anzugeben:

- SRCP: es folgt die SRCP Versionsnummer, die implementiert und unterstützt wird. Es muß eine der veröffentlichten Bezeichnungen sein!

Beispiel:

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

Anschliesend wartet der Server auf Kommandos des Client. Der Client muß den Informationstext lesen und kann nun seinerseits Kommandos an den Server schicken.

Der Rückmeldepollport wird nur unidirektional vom Server zum Client benutzt. Er dient der Information der Clients von Veränderungen an der Anlage.

Meldet sich ein Client am Rückmeldepollport an, so sendet der Server alle aktuell belegten Rückmelder und sodann jede Statusänderung einer Rückmeldeeinheit an den Client zurück. Mit diesem Mechanismus ist es möglich, Rückmeldungen beim Client ereignisgesteuert zu bearbeiten.

Auch der Informationsport wird nur unidirektional von Server zu den Clients benutzt. Er ist eine Art Broadcast-Kanal, der ständig vom Server mit Statusänderungen seitens anderer Clients oder möglicherweise vorhandener Informationsquellen der Anlage (nicht jedoch Rückmelder!) versorgt wird.

# 2 Kommandos zur Serversteuerung

Kommunikationsports

- **Client** -> **Server** Kommandoport
- **Server** -> **Client** entfällt

Kommandos

- **SHUTDOWN** Der Server beendet sich
- **LOGOUT** Dem Server wird angzeigt, dass sich ein Client ausloggt
- **RESET** Der Server re-initialisiert sich

Folgende Kommandos sind von Clients nicht mehr zu verwenden, müssen vom Server jedoch implementiert werden

- STARTVOLTAGE identisch mit SET POWER ON
- STOPVOLTAGE identisch mit SET POWER OFF

# 3 Kommandos zur Digitalsteuerung

## 3.1 Befehle und Gerätegruppen

SRCP definiert 8 generelle Befehle, die über den Kommandoport abgewickelt werden.

**SET**

Setzt einen Wert vom Client über den Server zum Gerät.

**GET**

Ermittelt den aktuellen Zustand eines Gerätes.

**WAIT**

Wartet, bis ein Gerät einen bestimmten Zustand erreicht hat.

**INIT**

Falls Geräte explizit initialisiert werden müssen.

**TERM**

Falls die mit INIT getroffenen Einstellungen wieder entfernt werden sollen.

**READ**

Liest einen Wert aus, Ergänzung zum WRITE

**WRITE**

Setzt einen Wert vom Client und liefert eine Antwort (Quittung) zurück.

**VERIFY**

Überprüft, ob ein Wert einen bestimmten Wert hat.

Geräte entstammen den Gerätegruppen Lokdekoder, Schaltdekoder, Rückmeldeeinheiten und sonstigen Bereichen.

Über Parameter wird festgelegt, auf welche Gerätegruppe sich ein Befehl bezieht. Der erste Parameter legt immer die Gerätegruppe fest:

Die Befehle WRITE, VERIFY und READ sind für die Verwendung von Programmierinterfaces (Dekoder) vorgesehen.

**GL**

Lok- und Funktionsdekoder (generic loco)

**GA**

Schaltdekoder (generic accessory)

**FB**

Rückmeldeeinheit (feedback)

**TIME**

Zeitnormal

**POWER**

Energieversorgung der Modellanlage

Anwendbarkeit der Befehle auf Gerätegruppen: Runde Klammern bedeuten optinale Elemente, die nicht von jedem Server unterstützt werden brauchen. Hierbei ist jedoch zu beachten, das z.B. WRITE zwar von jedem Server akzeptiert werden muß, dies jedoch nicht bedeutet, das er auch ausgeführt werden muß, wenn es an den Voraussetzungen fehlt. Hingegen darf TIME vollständig fehlen.

| | 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 | - | - | - | - | - | - |

Es ist zu beachten, das für die Gerätegruppen sowohl die Befehlsgruppen SET/GET als auch WRITE/READ spezifiziert sind. Die Parameterliste und auch der Verwendungszweck ist jedoch unterschiedlich!

Bei den Befehlen GET, WAIT, WRITE und READ erwartet der Client vom Server eine Antwort. Es kann nun vorkommen, daß der Server zur gestellten Anfrage keine Antwort geben kann. In diesen Fällen muß der Server eine Zeichenkette zum Client senden, die folgendem Format genügt:

```
INFO <error code>
```

Als "error code" muß eine negative Zahl übergeben werden. Folgende Festlegungen gelten:

**INFO -1**

==> Befehl wird nicht unterstützt (not supported)

**INFO -2**

==> Keine Information vorhanden (no data)

**INFO -3**

==> Zeitlimit überschritten (vgl. WAIT) (timeout)

# 4 Befehle für die Gerätegruppen

Für jede Gerätegruppe wird dargelegt, welche Befehle welche Wirkung erzielen sollen.

## 4.1 Generic Loco GL

Für die Steuerung von Lokdekodern und anderen mobilen Geräten mitsamt ihren Zusatzfunktionen.

### 4.1.1 SET GL

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

Kommunikationsports

- **Client** -> **Server** Kommandoport
- **Server** -> **Client** entfällt

Bedeutung der Argumente:

**protocol**

L, M, N, S, PS

- **L** Reserviert für Loconet
- **M1** Märklin alt (rel. FRU, 80 Adr., 1 Funkt., 14 FS)
- **M2** Märklin neu (abs. FRU, 80 Adr., 5 Funkt., 14 FS)
- **M3** M2 erweitert (abs. FRU, 256 Adr., 5 Funkt., 28 FS)
- **M4** M2 erweitert (abs. FRU, 256 Adr., 5 Funkt., 14 FS)
- **M5** M2 erweitert nach Märklin (abs. FRU, 80 Adr, 5 Funkt., 27 FS)
- **MF** altes Märklin Format für Funktionsdekoder
- **NB** NMRA-DCC Basisprotokoll (abs. FRU, 7-bit-Adr., 14 FS)
- **N1** NMRA-DCC erweitert (abs. FRU, 7-bit-Adr., 5/9/13 Funkt., 28 FS)
- **N2** NMRA-DCC erweitert (abs. FRU, 7-bit-Adr., 5/9/13 Funkt., 128 FS)
- **N3** NMRA-DCC erweitert (abs. FRU, 14-bit-Adr.,5/9/13 Funkt., 28 FS)
- **N4** NMRA-DCC erweitert (abs. FRU, 14-bit-Adr., 5/9/13 Funkt., 128 FS)
- **S** Reserviert für Selektrix
- **PS** protocol by server, der Server bestimmt den Protokolltyp

**addr**

Zahl >= 0

**direction**

0 (= rückwärts), 1 (= vorwärts), 2 (= Nothalt)

**V**

0 .. V_max ((virtuelle) Geschwindigkeit)

**V_max**

0 .. <pos.Zahl> (maximale (virtuelle) Geschwindigkeit) V_max=0 ==> virtuelle Fahrstufe = reale Fahrstufe

**func**

0 (= aus), 1 (= an)

**nro_f**

0 .. x (Anzahl der Zusatzfunktionen (number of functions))

**f1 .. fn**

0 (= aus), 1 (= an)

Ein Beispiel:

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

Die Lok mit einem erweiterten NMRA-DCC Decoder auf der Adresse 1 fährt mit 1/4 ihrer Maximalgeschwindigkeit vorwärts. Funktion und F2 sind aktiv.

Umrechnung der virtuellen Geschwindigkeit in echte Fahrstufen:

gegeben sind

- **DEC_fs** Anzahl der realen Dekoderfahrstufen, implizit bekannt
- **V** virtuelle Geschwindigkeit, Argument
- **V_max** maximale virtuelle Geschwindigkeit, Argument

gesucht

- **V_fs** reale Fahrstufe, die der virtuellen Geschw. entspricht

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

**Hinweise für Entwickler von SRCP-konformen Servern** Es ist darauf zu achten, daß wirklich nur dann die reale Fahrstufe 0 (Stillstand) errechnet wird, wenn das Argument V gleich Null ist. Die Funktion "round" ist deshalb hinreichend intelligent zu implementieren.

V_fs darf nur als Geschwindigkeitsangabe interpretiert werden. Manche Dekoder reagieren z.B. bei Fahrstufe 1 mit einem Nothalt, andere mit einem Richtungswechsel, wieder andere mit einer Selbstzerstörungssequenz ;-). Sollen solche Dekoder unterstützt werden, dann hat der Server dafür zu sorgen, daß V_fs entsprechend angepaßt wird, bevor das Kommando an den Dekoder gesendet wird. Aus Sicht der Clients müssen die Fahrstufen sukzessive von 0 bis zur maximalen Fahrstufenanzahl durchnummeriert sein.

Wird V_max auf 0 gesetzt, dann darf keine Umrechnung der Fahrstufe vorgenommen werden. D.h. die mit V übermittelte Fahrstufe ist direkt an die Dekoder weiterzusenden. Dies erlaubt Clients den direkten Zugriff auf die Dekoder.

Sendet der Client den Protokolltyp PS (protocol by server), dann muß der Server entscheiden, welches Protokoll er für die übermittelte Adresse wählt. Die anderen Protokolltypen stellen für den Server natürlich auch nur Empfehlungen des Clients dar. Der Server hat immer die Freiheit, einer Adresse ein anderes, geeigneteres Protokoll zuzuweisen.

Beispiele

```
(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>
```

Kommunikationsports:

- **Client** -> **Server** Kommandoport
- **Server** -> **Client** Kommandoport

Bedeutung der Argumente: siehe 4.1.1 (SET GL)

Der Server sendet an den Client alle verfügbare Info zu dem mit <protocol> und <addr> spezifizierten Lok- oder Funktionsdekoder. Diese Info muß wie folgt formatiert werden:

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

vgl. 4.1.1 (SET GL)

Sollte keine Information vorhanden sein, oder der Server den Befehl GET nicht unterstützen, dann muß der Server "INFO <error code>" an den Client senden.

## 4.2 Generic Accessoire GA

Mit den Generic Accessoires sind Schaltdekoder wie für Beleuchtung, Weichen und Signale erreichbar. Sie verfügen über getrennt ansprechbare Ports, die unterschiedliche Status annehmen können. Der Status 0 gilt als Grundzustand und kann vom SRCP-Server automatisch nach einer anzugebenden Verzögerungszeit gesendet werden.

#### 4.2.1 SET GA

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

Kommunikationsports:

- **Client** -> **Server** Kommandoport
- **Server** -> **Client** entfällt

Bedeutung der Argumente

**protocol**

- **M** Märklin/Motorola-Format
- **N** NMRA-DCC-Format
- **P** Protocol by Server: Der Server bestimmt den Protokolltyp

**addr**

Zahl >= 0 (Nummer der Weiche/des Signals)

**port**

0, 1, 2,... (Ausgang des Dekoders)

**action**

0, 1, 2, ... (0 => deaktiviert, >0 => aktiviert)

**delay**

Wert in Millisekunden (1000stel-Sekunden). Gibt an, nach welcher Zeit der Server einen aktivierten Ausgang automatisch deaktivieren (== auf 0 setzen) soll. Wird `"-1"` als delay übergeben, dann wird der Ausgang nicht automatisch deaktiviert. Ist action=0 (Deaktivierung) wird delay ignoriert, muß aber angegeben werden (sinnvoller Wert: `"-1"`).

Ein Port gilt als aktiv, wenn sein Status ungleich Null ist.

Belegung der Dekoderausgänge:

- **0** = Weiche abbiegen, Signal Hp0, ...
- **1** = Weiche gerade, Signal Hp1, ...

Beispiel:

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

Damit wird der Schaltausgang 23, Port 1 für 20 ms aktiviert. Achtung: bei einigen Systemen können Mindesteischaltzeiten existieren. Auch ist es möglich, daß die Ports einer Adresse nicht unabhängig voneinander geschaltet werden können. Oder daß keine zwei Adressen gleichzeitig aktiviert werden können.

### 4.2.2 GET GA

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

Kommunikationsports:

- **Client** -> **Server** Kommandoport
- **Server** -> **Client** Kommandoport

Bedeutung der Argumente: siehe 4.2.1 (SET GA)

Der Server sendet an den Client alle verfügbare Info über den aktuellen Zustand zu dem mit <protocol> und <addr> spezifizierten Schaltdekoder. Diese Info muß wie folgt formatiert werden:

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

(vgl. 4.2.1 (SET GA), ersetze <state> durch <action>)

Sollte keine Information vorhanden sein, oder der Server den Befehl GET nicht unterstützen, dann muß der Server `"INFO <error code>"` an den Client senden.

## 4.3 Zeitnormal TIME

Der Zeitgeber dient der Bereitstellung einer einheitlichen Modellzeit für alle Clients. Er ist ein optionales Feature, d.h. ein SRCP muß es nicht unterstützen.

### 4.3.1 INIT TIME

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

Startet den Zeitgeber mit der Verzerrung FX/FY am angegeben Zeitpunkt. Jede volle Modellminute wird sodann als INFO Paket auf dem INFO-Port aller aktiven und zukünftigen Clients ausgesendet.

Die Modellzeit errechnet sich wie folgt:

```
(Delta) Modellzeit = (Delta) Realzeit * FX / FY
```

Beispiele:

- FX=10 FY=1 -> Jede Realminute werden 10 Modellminuten generiert (also alle 6 Sekunden eine).
- FX=1 FY=10 -> Alle 10 Realminuten läuft eine Modellminute ab.
- FX=1 FY=1 -> Jede Realminute läuft eine Modellminute ab

Die Tageszahl wird fortlaufend alle 24 Modellstunden hochgezählt.

### 4.3.2 SET TIME

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

Kommunikationsports:

- **Client** -> **Server** Kommandoport
- **Server** -> **Client** entfällt

Bedeutung der Argumente

Die aktuelle Modellzeit und die Verzerrung gegenüber der Realzeit wird festgelegt. Bei erstmaligen Aufruf vergleichbar mit einem INIT TIME.

**JulDay**

sequentielle Folge von Tageszahlen (julianisch)

**Hour**

0..23, besteht aus 60 Minuten

**Minute**

0..59, besteht aus 60 Sekunden

**Second**

0..59

**FX, FY**

Ganzzahlige Bestandteile der Zeitverzerrung

Beispiel

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

setzt auf den Abend des ersten Tages mit Modellzeit gleich Realzeit. (vgl. 4.3.1 (INIT TIME))

### 4.3.3 GET TIME

```
GET TIME
```

Kommunikationsports:

- **Client** -> **Server** Kommandoport
- **Server** -> **Client** Kommandoport

Liefert die aktuelle Modellzeit und die Verzerrungsfaktoren als INFO-Zeile

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

Sollte keine Modellzeit definiert sein, so muß der Server dies mit "INFO -2" (no data) an den Client senden (vgl. 3.1 (Befehle und Gerätegruppen)).

### 4.3.4 WAIT TIME

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

Kommunikationsports:

- **Client** -> **Server** Kommandoport
- **Server** -> **Client** Kommandoport

Wartet, bis die Modellzeit den angegebenen Zeitpunkt mind. erreicht hat und liefert einen INFO-String mit der aktuellen Modellzeit.

bei nicht initialisiertem Zeitgeber wird "INFO -1" geliefert. Ist die aktuelle Modellzeit bereits später als die übergebende Zeit ist die Bedingung ohne weitere Wartezeit erfüllt. Offensichtlich falsche Zeitangaben werden durch "INFO -1" an den anfordernden Client ignoriert.

### 4.3.5 TERM TIME

Kommunikationsports:

- **Client** -> **Server** Kommandoport
- **Server** -> **Client** Kommandoport

Dieser Befehl stoppt den Zeitgeber. Er ist optional.

## 4.4 Energieversorung POWER

### 4.4.1 SET POWER

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

Kommunikationsports:

- **Client** -> **Server** Kommandoport
- **Server** -> **Client** entfällt

Bedeutung der Argumente

**State**

**ON**
Schaltet die Energieversorgung ein

**OFF**
Schaltet die Energieversorgung ab

**Freetext**

Ein optionaler Text mit maximal 100 Zeichen, der an das INFO Packet angehängt wird und nähere Informationen geben kann. Es werden ausdrücklich keine Vorgaben über den Inhalt gemacht.

### 4.4.2 GET POWER

```
GET POWER
```

Kommunikationsports:

- **Client** -> **Server** Kommandoport
- **Server** -> **Client** Kommandoport

Ermittelt den aktuellen Zustand der Energieversorgung. Als Antwort erscheint

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

<Freetext> ist hierbei der zuletzt gesetzte Wert oder ein vom Server anderweitig generierter Text. (vgl. 4.4.1 (SET POWER))

## 4.5 Rückmelder FB

### 4.5.1 INIT FB

```
INIT FB <module_type>
```

Kommunikationsports:

- **Client** -> **Server** Kommandoport
- **Server** -> **Client** entfällt

Der Server initialisiert das Rückmeldesubsystem. Damit stehen sowohl sowohl die Feedbackinformationen wie auch der FB-Port bereit. Keine Nachricht an den Client. Der Server kann den Befehl auch eigenständig ausführen (z.B. bei Programmstart), dann wird dieser Befehl ignoriert.

### 4.5.2 GET FB

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

Kommunikationsports:

- **Client** -> **Server** Kommandoport
- **Server** -> **Client** Kommandoport

Bedeutung der Argumente

**module_type**

- **S88** Märklin s88-Bus am Parallelport des PC
- **I8255** i8255 Karte
- **M6051** Märklin s88-Bus via Interface 6051
- **PS** Protocol by Server: Der Server entscheidet

**portnr**

konkreter Eingang eines Rückmeldemoduls oder `"*"` für alle verfügbaren Eingänge

Der Server sendet auf dem Rückmeldekanal an den Client den aktuellen Status des mit <module_type> und <portnr> spezifizierten Rückmeldemoduleingangs. Diese Info muß wie folgt formatiert werden:

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

Wobei `state` entweder `"0"` oder `"1"` sein darf.

Wurde als `portnummer` der Platzhalter `"*"` angegeben, so sendet der Server als portnummer den `"*"` (ohne Hochkomma), gefolgt von den Zuständen aller angeschlossenen Ports. Diese Zustände werden NICHT durch Leerzeichen getrennt. Zu beachten ist, das dieser String sehr lang werden kann!

Beispiel

```
INFO FB M6051 * 1100110010101111
```

Sollte keine Information vorhanden sein, oder der Server den Befehl GET nicht unterstützen, dann muß der Server `"INFO <error code>"` an den Client senden (vgl. 3.1 (Befehle und Gerätegruppen)).

### 4.5.3 WAIT FB

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

Kommunikationsports:

- **Client** -> **Server** Kommandoport
- **Server** -> **Client** Kommandoport

Wartet, bis der angegebene Port den Wert value (0,1) annimmt. Wartet aber höchstens `timeout` Sekunden. Sendet falls der timeout nicht eingetreten ist, die gleiche Information wie GET FB. Sollte der timeout überschritten werden, wird `"INFO -3"` an den Client gesendet.

Sollte keine Information vorhanden sein, oder der Server den Befehl WAIT nicht unterstützen, dann muß der Server `"INFO -1"` an den Client senden.

#### 4.5.4 TERM FB

```
TERM FB <module_type>
```

Kommunikationsports:

- **Client** -> **Server** Kommandoport
- **Server** -> **Client** entfällt

Der Server deaktiviert das entsprechende Rückmeldesubsystem. Dieser Befehl ist optional.

# 5 Kommandos zur Dekoderprogrammierung

Es gibt Dekoder, die auf einem Programmiergleis oder "on-track" programmierbar sind. Diese Dekoder erlauben die Änderung von "Speicherzellen" (Register, CV (configuration variable), Bit). Diese Dekoder können auch von SRCP-Servern unterstützt werden. Dazu werden die Befehle WRITE, VERIFY und READ spezifiziert.

In INFO Zeilen wird als Gerätegruppe das Wort SM benutzt: ServiceMode.

## 5.1 Dekodereinstellungen ändern

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

Kommunikationsports:

- Client -> Server: Kommandoport
- Server -> Client: Kommandoport

Bedeutung der Argumente

**protocol**

NMRA, ...

**dest-type**

Art der zu ändernden Dekoderspeicherzelle (destination type)

**CV**

NMRA-DCC configuration variable

**CVBIT**

ein Bit einer NMRA-DCC configuration variable, dest-addr enthält dann zwei Worte: Speicherzelle und Bitnummer (0-7) in dieser Speichezelle

**REG**

Ein Register eines NMRA-DCC-Dekoders

**dest-addr**

Adresse der Speicherzelle, die geändert werden soll (ggf. enthält sie die Bitnummer (0-7, bei CVBIT) als zweites Wort der Adresse.

**value**

neuer Wert der Speicherzelle

Ein SRCP-Server sendet immer nach dem Empfang und der Abarbeitung eines WRITE-Kommandos eine Information an den Client. Diese Information muss folgendes Format haben:

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

Wobei <value> folgende Werte annehmen kann:

- 0: WRITE ist fehlgeschlagen
- 1: WRITE wurde erfolgreich ausgeführt
- 2: Der Server weiss nicht, ob WRITE erfolgreich war

Sollte WRITE oder ein bestimmter Parameter von WRITE nicht vom Server unterstützt werden, so sendet der Server auf dem Kommandoport die Information "INFO -1".

## 5.2 Dekodereinstellungen verifizieren

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

Kommunikationsports:

- Client -> Server: Kommandoport
- Server -> Client: Kommandoport

Bedeutung der Argumente

**protocol**

NMRA, ...

**dest-type**

Art der zu verifizierenden Dekoderspeicherzelle (destination type)

**CV**

NMRA-DCC configuration variable

**REG**

Ein Register eines NMRA-DCC-Dekoders

**dest-addr**

Adresse der Speicherzelle, die verifiziert werden soll

**value**

zu verifizierender Wert der Speicherzelle

Ein SRCP-Server sendet immer nach dem Empfang und der Abarbeitung eines VERIFY-Kommandos eine Information an den Client. Diese Information muss folgendes Format haben:

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

Wobei <value> folgende Werte annehmen kann:

- 0: Der mit <value> angegebene Wert ist nicht der aktuelle Wert der Speicherzelle.
- 1: Der mit <value> angegebene Wert wurde verifiziert.
- 2: Der Server kann kein VERIFY durchführen.

Sollte VERIFY oder ein bestimmter Parameter von VERIFY nicht vom Server unterstützt werden, so sendet der Server auf dem Kommandoport die Information "INFO -1".

### 5.3 Dekodereinstellungen auslesen

Reserviert für zukünftige Erweiterungen.

## 6 Serverinformationsports

Die Serverinformationsports dienen der Information von Clients über Veränderungen der Anlage (Rückmelder) und die ausgeführten Befehle des SRCP Servers. Diese können jedoch nur als Hinweise für den Anlagenzustand gewertet werden.

### 6.1 Rückmeldeport

Die Daten müssen mit folgendem Format gesendet werden:

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

(vgl. 4.5.2 (GET FB))

Beim Öffnen des Ports werden sofort alle aktuell *belegten* (state == 1) FB-Ports an den Client übermittelt. Anschließend alle Veränderungen.

### 6.2 Informationsport

Die Daten müssen mit folgenden Formaten - abhängig von der Gerätegruppe - übertragen werden:

```
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
```

Andere als die oben angeführten dürfen nicht gesendet werden (insb. keine "INFO <error>"!).

Beim Öffnen des INFO Ports werden für jedes Gerät der Gerätegruppen GA und GL die INFO-Zeile der aktuelle Zustand, die aktuelle Modellzeit (TIME) und der Zustand von POWER an den Client gesendet.

Bei der Gerätegruppe GA besteht der aktuelle Zustand aus den aktuellen Zuständen der einzelnen Ports in ihrer zeitlichen Reihenfolge, in dem sie ihn eingenommen haben. Pro GA werden also ebensoviele INFO Zeilen gesendet, wie es Ports gibt.

Bei der Gerätegruppe GL ist dies der dem letzten SET Befehl entsprechende INFO String, sofern der Server nicht über andere, anlagenbezogene Informationsquellen verfügt.

Hierbei werden nur alle bereits benutzten Geräte bzw. die, deren Zustand der Server sicher ermitteln kann (z.B. durch Abfragen der anlagenseitigen Geräte, sofern die dies unterstützen), verwendet und nicht der gesamte Adreßraum übertragen!

Danach werden alle Veränderungen der verfügbaren Geräte übermittelt, sobald sie das betreffende Gerät erreicht haben bzw. der Server sie ermitteln konnte.

# 7 Ausblicke, zukünftige Erweiterungen

Einige Themen wurden und werden in der Diskussion zwar angesprochen, haben jedoch keinen Eingang in die vorliegende Fassung von SRCP gefunden. Damit sie trotzdem nicht verloren gehen und für zukünftige Erweiterungen "frisch bleiben" sind sie nachfolgend angeführt. Sie sind also ausdrücklich nicht Bestandteil des Simple Railroad Command Protocols.

## 7.1 Zentrale Konfiguration

Von Kurt Haders stammt der Vorschlag einer zentralen Konfigurationsdatei. Hauptanliegen ist es, mehr "Wissen" von den Clients in den Server zu verlagern. Ein Format, wie eine solche Konfigurationsdatei aussehen soll, liegt noch nicht vor. Das Übertragungsprotokoll ist duch den Protokollbezeichner "PS" (protocol by server) bereits darauf vorbereitet.

## 7.2 Erweiterung der Befehle auf andere Gerätegruppen

Von Matthias Trute kommt der Vorschlag den Befehl WAIT auch für die Gerätegruppen GL und GA zuzulassen. Allerdings kann es hierbei zu Inkonsistenzen kommen. Weiterhin könnte man man bei WAIT Wildcards ("*") zulassen.

## 7.3 Quittierung von Befehlen

Martin Ostermann hängt an einer Befehlsquittierung. Dieses könnte über den Rückkanal des Kommandoports realisiert werden.

## 7.4 Konfiguration des Servers auslesen

Von Edbert van Eimeren kommt der Vorschlag, daß Clients die Konfiguration des Servers abfragen können sollten (neuer Befehl: CONFGET).

### 7.5 Modularisierung

Es gibt verschiedene Ideen, über das Protokoll Module anzusprechen, die bestimmte Sonderfunktionen erreichen.

### 7.6 Sperren einzelner Geräte

Von Martin Wolf kommt die Idee, einen optionalen Lock-Mechanismus vorzusehen. Darüber kann ein Client für eine laufende Sitzung ein Gerät (GA, GL, FB, POWER,...) für sich reklamieren. Wenn sich der Client ausloggt (oder die Verbindung mit ihm abbricht) wird der Lock aufgehoben. Dies könnte über die Modularisierung des Protokolls abgebildet werden. Hierfür sind neue Befehle möglich: LOCK, UNLOCK, BREAKLOCK. Oder ein Einsatz der bisherigen: SET LOCK 1, SET LOCK 0, WAIT LOCK, GET LOCK etc.

## 8 Glossar

Hier werden einige Begriffe erklärt, die nicht unbedingt Allgemeingut sind aber eine tragende Rolle im SRCP spielen.

**Julianische Tageszahl**

Laufende Tageszählung anstelle eines strukturierten Kalenders. Ausgehend von einem festgelegten Starttag und -zeitpunkt wird alle 24 Stunden der folgende Tag begonnen. Gebräuchlich sind als Startzeitpunkt wiederkehrend Neujahr 0h:0:0 (wie in der C Runtime) bzw. einmalig der 1.1.4713 v.Chr. 12-00 GMT in der Astronomie.

Es existieren Algorithmen zur Umrechung von Kalenderinformationen in diese Darstellungsform und umgekehrt (z.B. in Collected Algorithms from CACM #199)