

rcsh – Railroad Command Shell

Dr. Peer Griebel

Inhalt

<u>rcsh – Railroad Command Shell</u>	1/20
<u>Copyright und Lizenz</u>	1/20
<u>Einführung</u>	2/20
<u>rcsh</u>	2/20
<u>Exkurs: rman</u>	2/20
<u>... zurück zu rcsh</u>	2/20
<u>Ein paar Worte über SRCP</u>	2/20
<u>Ein paar Worte über Python</u>	3/20
<u>Systemanforderungen</u>	4/20
<u>rcsh – Railroad Command Shell</u>	6/20
<u>1. Verwaltung/Steuerung des Gesamtsystems</u>	6/20
<u>2. Steuerung von Lokomotiven</u>	8/20
<u>3. Steuerung von Weichen/Signalen usw.</u>	9/20
<u>4. Zeitgesteuerte Ansteuerung von Lokomotiven/Weichen</u>	10/20
<u>5. Steuerung durch Rückmelder (synchron)</u>	11/20
<u>6. Steuerung durch Rückmelder (asynchron)</u>	13/20
<u>Weiterführende Informationen</u>	16/20
<u>Appendix A – GNU General Public License</u>	A-1

rcsh – Railroad Command Shell

Diese Dokumentation beschreibt *rcsh* Version 1.0 (2002–06–03).

Die neueste Version der software und der Dokumentation ist zu finden unter <http://www.griebel-net.de/peer/rcsh.html>.

Dieses Dokument soll eine Dokumentation zur *rcsh* bieten, die zumindest einem durchschnittlich versierten Anwender die Möglichkeit gibt, die *rcsh* unter Linux-Systemen erfolgreich einzusetzen. (*rcsh* ist allerdings grundsätzlich auf allen Systemen einsetzbar, auf denen das Python-System (ggf. inklusive Threads) verfügbar ist.) Sollte die Dokumentation Lücken oder Fehler aufweisen oder liegen einfach Verbesserungsvorschläge für *rcsh* und seine Dokumentation vor, so würde ich mich sehr darüber freuen, eine entsprechende [Nachricht](#) zu erhalten.

Copyright und Lizenz

rcsh und *rcman* mit allen Komponenten und Dokumentation: ©2000–2002 by Dr. Peer Griebel (peer.griebel@web.de).

Dieses Programm ist freie Software. Sie können es unter den Bedingungen der [GNU General Public License](#), wie von der [Free Software Foundation](#) herausgegeben, weitergeben und/oder modifizieren, entweder unter Version 2 der Lizenz oder (wenn Sie es wünschen) jeder späteren Version. Die Veröffentlichung dieses Programms erfolgt in der Hoffnung, dass es Ihnen von Nutzen sein wird, aber OHNE JEDE GEWÄHRLEISTUNG – sogar ohne die implizite Gewährleistung der MARKTREIFE oder der EIGNUNG FÜR EINEN BESTIMMTEN ZWECK. Details finden Sie in der GNU General Public License.

Einführung

rcsh

rcsh dient dazu, eine komfortable und vollständige Schnittstelle zu [SRCP](#) anzubieten. *rcsh* ist in der Programmiersprache [Python](#) geschrieben. Der Anlass, *rcsh* zu schreiben ergab sich, als mir klar wurde, dass *ddsh* von Torsten Vogt nicht mehr weiter gepflegt werden würde.

Ich selbst bin ein großer Fan von textuellen Shells. Außerdem wollte ich mein altes Notebook zur Steuerung der Modelleisenbahnsteuerung weiter verwenden. Daher ist für mich die Verwendung einer grafischen Oberfläche ausgeschieden.

Exkurs: rcman ...

Mittlerweile habe ich neben dem Programm *rcsh* das Programm *rcman* recht weit gebracht. Dokumentation zu *rcman* ist allerdings ein anderes Thema ;^) *rcman* unterstützt derzeit das Steuern von Lokomotiven und das Stellen von Weichenstraßen. Die Bedienung erfolgt textuell, die Informationen über die Loks und Weichenstraßen werden jedoch recht ansprechend durch Zeichengrafik dargestellt. Mir persönlich gefällt besonders gut, dass das Programm über eine kabellose Wheel-Mouse gesteuert werden kann! Das Rad steuert die Geschwindigkeit, die Maustasten dienen zur Steuerung der verschiedenen Lokfunktionen und zur Anwahl der verschiedenen Loks. Eine Steuerung per Infrarot-Schnittstellen ist ebenfalls vorgesehen...

... zurück zu rcsh

Doch zurück zu *rcsh*. Sehen wir uns einfach ein kleines Beispiel an, um einen Eindruck von *rcsh* zu erhalten:

```
from rcsh import * # importiere die rcsh-Grundfunktionen

Connect()         # stelle eine Verbindung zum Server her
Power()           # schalte Strom auf den Schienen ein
lok = locoM2(34)  # definiere ein Lok
lok.setSpeed(5)   # setze die Geschwindigkeit
lok.send()        # sende die Geschwindigkeit an die Lok
```

Obige Zeilen können direkt am Kommandoprompt eingegeben werden, nachdem Python gestartet wurde. (Die Kommentare beginnend mit dem Lattenzaun '#' müssen nicht eingegeben werden.) Die Zeilen definieren eine Lok mit dem neuen Märklin Protokoll mit Adresse 34. Die Geschwindigkeit der Lok wird auf 5 gesetzt und diese Information an die Lok selbst gesendet. Das Beispiel ist bewusst einfach gehalten. Es soll lediglich einen ersten Eindruck vermitteln. Die nächsten Abschnitte stellen nun den Umgang mit *rcsh* im Detail dar. Dazu wird zunächst ein kleiner Blick auf [SRCP](#) und Python geworfen, bevor der Leistungsumfang von *rcsh* selbst beleuchtet wird.

Ein paar Worte über SRCP

Dieser Abschnitt ist für das Verständnis von *rcsh* nicht unmittelbar notwendig. Es kann auch mit dem Lesen des folgenden Abschnitts fortgefahren werden.

Aus der Dokumentation von SRCP:

SRCP (Simple Railroad Command Protocol) *"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."*

SRCP beschreibt also den Austausch von Kommandos und Rückmeldungen zwischen *rcsh* und einem Server. Die Kommunikation zwischen Server und *rcsh* erfolgt normalerweise im Hintergrund und ist normalerweise vor dem Anwender verborgen. Um jedoch detaillierter zu wissen, was hinter den Kulissen vor sich geht, oder aber um vertrackte Probleme zu debuggen, empfiehlt es sich, sich etwas mehr mit dem SRCP zu befassen. SRCP liegt unter <http://www.der-moba.de/Digital>.

Ein paar Worte über Python

Python ist eine objektorientierte Programmiersprache (Skriptsprache). Python–Programme werden nicht übersetzt im eigentlichen Sinn, sondern werden interpretiert. Python–Programme sind sehr kompakt und dennoch gut lesbar. Python ist für viele Betriebssysteme verfügbar (auch für Windoof). Viele Linux–Distributionen beinhalten Python, es muss jedoch gezielt installiert werden. Es empfiehlt sich, die Dokumentation von Python zumindest grob zu überfliegen, um die Sprache erfolgreich einzusetzen. Hier können nur die wichtigsten Dinge kurz und knapp vorgestellt werden.

Der augenfälligste Unterschied zwischen Python und anderen Programmiersprachen ist, dass es in Python keine Klammern gibt, um Programmblöcke zusammen zu halten. Zu diesem Zweck dient lediglich die Einrückung. In folgendem Programmfragment ist durch Einrückung eindeutig definiert, dass die Anweisungen A1 und A2 durchlaufen werden, wenn die if–Abfrage positiv ausfällt, die Anweisung A3 und A4 im negativen Fall:

```
if Bedingung:
    A1
    A2
else:
    A3
    A4
```

Python kann sehr gut interaktiv bedient werden. Sie können Python starten und beispielsweise das Beispiel aus der Einführung eingeben. Python bietet ein komfortables Editieren der Kommandozeile. Somit lassen sich alte Eingaben einfach wiederholen oder abändern. Objekte lassen sich inspizieren, einfache Schleifen o.ä. realisieren. Der Interpreter wird verlassen, indem Ctrl–D (Ctrl–Z unter der Windoof–Version) gedrückt wird.

Komplexere Skripte, die mehrfach verwendet werden sollen, sollten in eigenen Dateien mit der Endung `.py` abgelegt werden. Das heißt, das Beispiel aus der Einleitung kann in der Datei `demo1.py` abgelegt werden. Der Aufruf dieses Skripts erfolgt dann mittels

```
python demo1.py
```

Um nicht immer den Kommandointerpreter `python` nennen zu müssen, bietet Linux (bzw. Unix generell) einen speziellen Mechanismus an. Hierfür muss in die erste Zeile, erste Spalte der Text ähnlich dem folgenden stehen:

```
#!/usr/bin/python
```

Diese Zeile besagt, dass das aktuelle Skript mit dem Programm python im Verzeichnis /usr/bin zu starten ist. Für Ihre Umgebung ist dieses Skript ggf. anzupassen und der richtige Pfad bzw. der richtige Name des Programms anzugeben.

Immer wieder kehrende, in Python realisierte Funktionalitäten können in eigene Dateien (Paket) ausgelagert werden. Ein Skript kann auf die Funktionalität zurückgreifen, indem das Paket importiert wird. Importieren geschieht auf zwei verschiedene Arten. Mittels der Anweisung

```
from paket import *
```

wird paket.py importiert. Die Funktionalitäten stehen unmittelbar zur Verfügung. Eine Funktion mit dem Namen fkt kann somit direkt mit fkt() aufgerufen werden. Dem gegenüber importiert die Anweisung

```
import paket
```

das Paket in einem eigenen Namensraum. Alle Definitionen aus paket stehen zur Verfügung, indem die Zeichenkette paket. vorangestellt wird. Die Funktion fkt wird also durch paket.fkt() aufgerufen.

Das Paket *rcsh* wird üblicherweise durch die erste Methode (`from rcsh import *`) importiert. Es wird also nicht vor jedes *rcsh*-Kommando `rcsh.` geschrieben, was gerade bei der interaktiven Bedienung hinderlich wäre.

Noch ein Wort zum Aufruf von Python. Python kennt den Kommandozeilenschalter `-O`. Mit diesem Schalter optimiert Python das Programm in geringem Maße. Im Falle von *rcsh* bedeutet dies, dass alle Debug-Anweisung vollständig ignoriert werden und daher das Programm etwas schneller läuft. Gerade auf langsameren Rechnern kann dies in kritischen Situationen vorteilhaft sein. Im Allgemeinen fallen diese Debug-Anweisungen jedoch nicht ins Gewicht.

Python kennt Default-Argumente und benannte Argumente für Funktionen und Methoden. Von diesem Vorzug wird in *rcsh* regen Gebrauch gemacht. Die Funktion `connect` beispielsweise ist definiert als

```
def connect(host='localhost', port=12345)
```

Das heißt die Funktion akzeptiert zwei Parameter, wobei beide Parameter mit Default-Werten belegt sind. Wird ein Parameter also nicht angegeben, so greift der Default-Wert. Dies wird an Beispielen deutlich. `connect()` erstellt eine Verbindung mit dem SRCP-Server auf dem aktuellen Rechner (`localhost`) und mit der Portnummer 12345. `connect('some.other.host')` erstellt eine Verbindung mit dem SRCP-Server auf dem Rechner `some.other.host` aber mit der Portnummer 12345. Benannte Argumente erlauben das Konstrukt `connect(port=23456)`. Mit dieser Anweisung wird eine Verbindung mit dem SRCP-Server auf der Lokalen Maschine (`localhost`) herzustellen, allerdings unter der Portnummer 23456.

Systemanforderungen

rcsh in der Version 0.6 erwartet Python 1.x. Die Python Version 2.x wird erst ab *rcsh* Version 0.7 unterstützt. *rcsh* benötigt weiterhin einen Server nach SRCP-Spezifikation.

Bisher wird ausschließlich SRCP in Version 0.7 unterstützt.

rcsh – Railroad Command Shell

rcsh wurde getestet unter den Betriebssystemen Linux und Windows. Unter letztem Betriebssystem wurden die CygWin Utilities (www.cygwin.com) verwendet, die Python 2 beinhalten.

rcsh – Railroad Command Shell

Die *rcsh* teilt sich in mehrere unabhängige Teile, die nun nachfolgend einzeln vorgestellt werden sollen:

1. Verwaltung/Steuerung des Gesamtsystems
2. Steuerung von Lokomotiven
3. Steuerung von Weichen/Signalen usw.
4. Zeitgesteuerte Ansteuerung von Lokomotiven/Weichen
5. Steuerung durch Rückmelder (synchron)
6. Steuerung durch Rückmelder (asynchron)

1. Verwaltung/Steuerung des Gesamtsystems

Unter diesen Bereich fallen die allgemeinen Vorkehrungen, die getroffen werden müssen, um mit dem SRCP-Server kommunizieren zu können und um den Server steuern zu können. Der Mechanismus ist objektorientiert implementiert. Um jedoch einen möglichst einfachen und intuitiven Zugang zu ermöglichen, sind spezielle Funktionen vorgesehen. Zunächst sollen diese speziellen Funktionen vorgestellt werden, bevor auf die objektorientierte Implementation eingegangen wird, die im Hintergrund steht. Für den direkten Zugang zum SRCP-Server und die Steuerung desselben stehen folgende Funktionen zur Verfügung.

`Connect` ist üblicherweise eine der ersten Anweisungen, die bei einem Programm aufgerufen wird, das auf *rcsh* aufbaut. Die Funktion stellt eine Verbindung zum SRCP-Server mit den spezifizierten Angaben (`host` und `port`) her. Sollte es Probleme beim Verbindungsaufbau geben, so wird eine entsprechende Exception geworfen. Die Verbindung geht in diesem Fall in den Offline-Modus. Die Exception kann im aufrufenden Programm aufgefangen werden, um im Offline-Modus arbeiten zu können. Der Offline-Modus ermöglicht es, *rcsh* zu verwenden, um die prinzipielle Funktionsweise eines Programms zu testen, ohne Befehle wirklich an den SRCP-Server zu versenden. Da jedoch keine Antworten vom SRCP-Server kommen, sind nur einfache Tests möglich.

```
def Connect(host='localhost', port=12345):
    """Establish a connection to the SRCP-server

    host and port specify the parameters of how to connect
    the server. If the connection could not be established
    the according exception will be raised."""
```

Die Digitalspannung am Gleis kann mit der Funktion `Power()` an- und abgeschaltet werden.

```
def Power(p=1):
    """Enable/disable power on the tracks"""
```

Den aktuellen Zustand über die Versorgung der Gleise mit Digitalspannung kann mit der Funktion `GetPower()` abgefragt werden:

```
def GetPower():
    """Determine the current setting of the power switch"""
```

Mit der Funktion `Reset()` kann eine Neu-Initialisierung des SRCP-Server veranlasst werden.

```
def Reset():
    """Re-initialize the server"""
```

Als letztes steht die Funktion `Monitor()` zur Verfügung, die die Analyse und Auswertung der Daten auf dem Info- und Rückmeldeport erlaubt. Die genaue Funktionsweise wird etwas später im Zusammenhang mit der objektorientierten Kapselung des SRCP-Servers besprochen.

Wie erwähnt stellen die oben genannten Funktionen nur eine Abkürzung zur objektorientierten Implementation für den Zugang zum SRCP-Server bereit. Die Klasse, die hinter o.g. Funktionen steht, heißt `srcp_connection`. `rcsh` instanziiert automatisch ein Objekt dieser Klasse mit dem Namen `srcp`. Die Funktion `Connect` beispielsweise stützt sich auf dieses Objekt ab. Die Implementation von `Connect` lautet:

```
def Connect(host='localhost', port=12345):
    srcp.connect( host, port )
```

Mit anderen Worten `Connect` delegiert die Aufgabe, eine Verbindung herzustellen, einfach an die Methode `connect` des Objekts `srcp`. Entsprechend sind die übrigen Funktionen implementiert.

Nun können wir uns auch mit der Aufgabe der Methode `monitor()` bzw. der bereits erwähnten Funktion `Monitor()` beschäftigen. `srcp_connection.monitor()` dient (indirekt) dazu, Informationen, die auf dem Info- bzw. dem Rückmeldepollport ankommen, auszuwerten und zu verarbeiten:

```
def monitor(self, infoport, pollport=None)
```

Die Funktion `monitor` bereitet die eigentliche Verarbeitung nur vor, indem der `infoport` und — falls gewünscht — der `pollport` geöffnet werden. Anschließend wird ein separater Thread gestartet, der die eigentliche Verarbeitung übernimmt. Konkret werden derzeit zwei Aufgaben übernommen. Einerseits werden Zustandsänderungen an den Lokomotiven (INFO GL, siehe SRCP) und der Accessories (INFO GA) ausgewertet. Diese Änderungen werden an die definierten Lokomotiven (siehe [Steuerung von Lokomotiven](#)) bzw. Weichen ([Steuerung von Weichen](#)) weitergereicht. Somit haben die Lokomotiv- bzw. Weichenobjekte jeweils aktuelle Informationen über die aktuellen Einstellungen, wie sie der Server selbst pflegt. Andererseits werden durch den Monitor-Thread Statusänderungen an den Feedback Devices (INFO FB, siehe SRCP) ausgewertet. Interessante Ereignisse werden an entsprechend registrierte Funktionen (Handler) weitergereicht, siehe [Steuerung durch Rückmelder \(asynchron\)](#).

Bei der Verwendung der Funktion `Monitor()` ist noch eine Besonderheit zu beachten. Da für die Funktion ein separater Thread verwendet wird, der einige Zeit benötigt, um gestartet zu werden, und es auch eine gewisse Zeit braucht, bis alle Informationen vom SRCP-Server über die Loks, Weichen und FBs verarbeitet sind, verstreicht eine gewisse Zeit, bis wirkliche Änderungen am System mitgeteilt und verarbeitet werden. Dies muss ggf. bei der Programmierung berücksichtigt werden. Die nachfolgend vorgestellten Lok- und Weichenobjekte initialisieren sich jedoch selbständig, so dass unmittelbar der korrekte, aktuelle Zustand des Servers verfügbar ist.

Üblicherweise ist es nicht notwendig eigene Instanzen der `srcp_connection` Klasse zu erzeugen. Eine Ausnahme ergibt sich in der Situation, wenn mehrere unabhängige Threads erzeugt werden sollen, die jeweils eine eigene Verbindung zum SRCP-Server benötigen. In

diesem Fall erhält jeder Thread sein eigenes `srcp`-Objekt. Die im weiteren vorgestellten Klassen akzeptieren alle einen optionalen Parameter im Konstruktor, bei dem das jeweils gültige `srcp`-Objekt übergeben werden kann. Somit ist es möglich, Loks, Weichendekoder usw. gezielt einem Thread und damit einer `srcp_connection` zuzuweisen.

2. Steuerung von Lokomotiven

Entsprechend dem SRCP-Standard unterstützt *rcsh* eine Vielzahl von verschiedenen Lokomotiv-Dekodern:

- ◆ 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)
- ◆ NB: NMRA-DCC Basisprotokoll (abs. FRU, 7-bit-Adr., 14 FS)
- ◆ N1: NMRA-DCC erweitert (abs. FRU, 7-bit-Adr., 5 Funkt., 28 FS)
- ◆ N2: NMRA-DCC erweitert (abs. FRU, 7-bit-Adr., 5 Funkt., 128 FS)
- ◆ N3: NMRA-DCC erweitert (abs. FRU, 14-bit-Adr., 5 Funkt., 28 FS)
- ◆ N4: NMRA-DCC erweitert (abs. FRU, 14-bit-Adr., 5 Funkt., 128 FS)

(Auch der Dekodertyp PS – protocol by server ist vorgesehen. Ich habe ihn allerdings noch nicht getestet. Ich muss gestehen, ich weiß nicht einmal was ihn auszeichnet...) Jedem der oben genannten Dekodertypen ist in *rcsh* eine Klasse zugeordnet: `locoM1`, `locoM2`, ..., `locoN4`, `locoPS`. Der Konstruktor der Klasse `locoXX` hat folgende Signatur:

```
def __init__(self, addr, srcp=srcp)
```

Durch die Instanzierung des Lok-Objekts wird automatisch der aktuelle Zustand der Lok mit dieser Adresse vom Server erfragt. Dieser Zustand steht demnach sofort bereit. Eine Lok wird am einfachsten definiert durch den Aufruf `locoXX(addr)`. `XX` ist zu ersetzen durch den jeweiligen Dekodertyp; `addr` steht für die Adresse des jeweiligen Dekoders in der Lok. Das folgende Beispiel definiert demnach eine Lokomotive mit neuem Märklin-Dekoder mit der Adresse 34:

```
lok = locoM2(34)
```

Als optionaler Parameter ist ein `srcp`-Objekt anzugeben, um bei Verwendung mehrerer Threads jedem Thread ein `srcp`-Objekt zuzuweisen und mit diesem Objekt wiederum die Loks zu verbinden.

Das Objekt `lok` ist nun vorbereitet, Steuerinformationen entgegen zu nehmen. Zu beachten ist, dass jede Steuerinformation zunächst in dem Objekt `lok` gesammelt wird, bis alle Einstellungen gesammelt an die wirkliche Lok auf dem Gleis gesendet wird. Dazu dient der Befehl `send()`. Die Klassen `locoXX` verstehen folgende Methodenaufrufe [ACHTUNG: Die Methodennamen und die Signaturen haben sich zwischen den Versionen 0.7 und 0.8 geändert. Es sollten nur noch die neuen Methoden verwendet werden. Die alten Methoden sind jedoch bis auf weiteres als deprecated markiert verfügbar.]:

```
def setSpeed(self, s):
    """set speed setting"""
def getSpeed(self):
    """get speed setting"""
def getNumFunctions(self):
    """get number of supported functions by decoder"""
def getFunction(self):
    """get setting of FUNCTION"""
def setFunction(self,v):
```

```

    """set/get setting of FUNCTION"""
def toggleFunction(self):
    """toggle setting of FUNCTION"""
def setF(self, n, v):
    """set Fn to v"""
def getF(self, f):
    """get value of Fn"""
def getFList(self):
    """get list of all Fn (incl. Func)"""
def setFList(self, v):
    """set list of all Fn (incl. Func)"""
def toggleF(self, n):
    """toggle value of Fn"""
def getDecodertype(self):
    """get decodertype e.g. 'M2'"""
def getDecoderDescription(self):
    """get decoder name e.g. 'M2 Maerklin new'"""
def getAddress(self):
    """get address of decoder"""
def getDecoderMaxSpeed(self):
    """get maximum speed level supported by decoder"""
def setMaxSpeed(self, s):
    """set maximum virtual speed"""
def getMaxSpeed(self, s = None):
    """get maximum virtual speed"""
def getDirection(self):
    """get direction"""
def setDirection(self, d):
    """set direction"""
def toggleDirection(self):
    """toggle direction"""

```

Die Aufgaben der Funktionen sollte selbsterklärend sein [Wenn nicht, bitte ich um eine kurze Nachricht!].

Der aktuelle Zustand einer Lok wird bei einer Statusabfrage grundsätzlich nicht beim Server erfragt. Stattdessen pflegt jedes locoXX-Objekt die aktuellen Zustände selbst. D.h. Eine Abfrage eines Zustands wird immer den zuletzt geschriebenen Wert zurück liefern. Greift nur ein Client auf den SRCP-Server zu, so ist diese Situation ausreichend. Der Client setzt Zustände und kann sie später auch wieder abfragen. Komplexer wird es, wenn mehrere Clients auf die gleiche Lok zugreifen. Dann kann es sein, dass ein Client den Zustand der Lok verändert und der andere Client darüber benachrichtigt werden muss. Dazu dient die Methode `monitor` der Klasse `srcp_connection` (bzw. die Funktion `Monitor`), wie sie bereits in [Verwaltung/Steuerung des Gesamtsystems](#) vorgestellt wurde. In diesem Fall teilt der SRCP-Server Zustandsänderungen automatisch an das `srcp`-Objekt mit, das mittels der Methode `monitor()` diese Zustandsänderungen an das jeweilige Lok-Objekt weiter reicht. Nun hat das Lok-Objekt immer den im Server aktuellen Zustand gespiegelt; Abfragen über den Zustand liefern grundsätzlich den aktuellen Stand zurück.

3. Steuerung von Weichen/Signalen usw.

Die Ansteuerung von Weichen erfolgt ähnlich wie die der Lokomotiven. Zunächst ist ein Objekt für eine Weiche zu instanzieren. Es existieren zwei Typen von Weichendekodern: Märklin- und NMRA-Dekoder. Entsprechend existieren die Klassen `turnoutM` und `turnoutN`. Die Dekoder können auch mit den Klassen `signalM` bzw. `accessoryM` und `signalN` bzw. `accessoryN` instanziiert werden. Letztere Klassen sind einfach andere Namen für die erstgenannten (es gilt: `turnoutM = signalM = accessoryM`, bzw. `turnoutN = signalN = accessoryN`). Der Konstruktor der Klasse `turnoutXX` hat folgende Signatur:

```
def __init__(self, addr, delay=-20, srcp=srcp)
```

Die Parameter `addr` und `srcp` entsprechen den Parametern wie bei [locoXX](#). Wie auch bei `locoXX` wird durch Instanziierung eines `turnoutXX`-Objekts dessen aktueller Zustand beim Server erfragt und steht sofort zur Verfügung. Der Parameter `delay` wird nachfolgend durch die Methode `setDelay()` beschrieben. Die Klasse `turnoutXX` kennt folgende Methoden:

```
def getDecodertype(self):
    """get decodertype e.g. 'M'"""
def actuate(self, port, action=1):
    """activate the accessory

    An accessory has two ports (0 and 1). actuate activates
    (action=1) or deactivates (action=0) one of these ports.
    Normally deactivation is done automatically by the srcp-server
    of by this method actuate (see setDelay).
    """
def setDelay(self, newDelay):
    """Set the delay to automatically switch off the actuator activation

    The time is given in milli-seconds.
    Positive values delegate the treatment to the SRCP-server.
    Negative values result in a local handling: The accessory waits
    itself until the time passed and afterwards switches off the
    actuator command.
    The special value None is used to disable autmatic switch off.
    """
def hp0(self):
    """a shortcut for self.actuate(0)"""
set = hp0
def hpl(self):
    """a shortcut for self.actuate(1)"""
reset = hpl
def toggle(self):
    """toggles the current state of the accessory"""
def getState(self):
    """returns the current state of the accessory"""
def getAddress(self):
    """returns the address of the accessory"""
```

Um eine Weiche/ein Signal anzusteuern existieren die Methoden `set()` und `reset()` (und dazu gleichbedeutende Methoden `hp0()` und `hpl()`). im Gegensatz zu Lokomotivdekodern ist kein explizites `send()` nötig – im Gegenteil: es ist gar nicht möglich. `set()` bzw. `reset()` wird direkt an das Gerät (Weiche o.ä.) gesendet. Das nachfolgende Beispiel verdeutlicht die Benutzung:

```
signal = signalM(11)
weiche = turnoutM(12)
signal.set()
weiche.reset()
```

4. Zeitgesteuerte Ansteuerung von Lokomotiven/Weichen

Mit den bis hierher vorgestellten Möglichkeiten von `rcsh` und Python ist es möglich, Loks und Weichen zeitgesteuert zu kontrollieren. Nachfolgend vorgestellte Funktion definiert einen zeitlichen Ablauf einer Ansteuerung einer Lokomotive. Neu an dem Code ist die Verwendung der Funktion `time.sleep()`. Diese Funktion tut einfach eine gewisse Zeit gar nichts. Sie verzögert lediglich den weiteren Ablauf der restlichen Funktionalität. Der Parameter zur Funktion `sleep()` spezifiziert die Zeit in Sekunden. Neu ist weiterhin, dass eine Folge von Anweisungen zu einer Funktion — hier `Zug1` — zusammengefasst werden kann:

```
#!/usr/bin/python

from rcsh import *

# Definition von Zug 1 (E94, DCC-Decoder)
def Zug1():
    l = locoN1(4)
    # Licht an, langsam aus Bahnhof herausfahren
    l.setFunction()
    l.setSpeed(1)
    l.send()
    # Immer noch langsam im Bahnhofsbereich
    l.setSpeed(3)
    l.send()
    time.sleep(12)
    # Beschleunigen auf freier Strecke
    l.setSpeed(7)
    l.send()
    time.sleep(20)
    # langsam in Bahnhof
    l.setSpeed(2)
    l.send()
    time.sleep(5)
    # Kriechen und Stillstand an alter Startposition
    l.setSpeed(1)
    l.send()
    time.sleep(3)
    l.setSpeed(0)
    l.setFunction(0)
    l.send()
    print l.getSpeed()

# starte die Verarbeitung
Zug1()
```

Obige Funktion definiert also den zeitlichen Ablauf für eine Lokomotive. Nun fährt auf einer Anlage normalerweise nicht nur eine Lokomotive. Daher bietet es sich an, mehrere Funktionen nach dem obigen Schema zu definieren, die auch andere Lokomotiven Steuern. Da die Lokomotiven gleichzeitig fahren sollen, müssen sie natürlich auch gleichzeitig gesteuert werden. Die entsprechenden Funktionen `Zug1`, `Zug2` usw. müssen also gleichzeitig ausgeführt werden. Dies kann erreicht werden, indem für jede Lok, also für jede der drei Funktionen ein Thread gestartet wird. Dies wird erreicht mit folgender Code-Sequenz:

```
import threading

threading.Thread(target=Zug1).start()
threading.Thread(target=Zug2).start()
threading.Thread(target=Zug3).start()
```

Die Threads werden jeweils genau einmal durchlaufen und terminieren anschließend. Sollen die Lokomotiven mehrmals fahren, so ist eine geeignete Schleife um den Start der Threads zu programmieren.

5. Steuerung durch Rückmelder (synchron)

Die einfachste Art und Weise, auf Rückmelder zu reagieren, ist, aktiv auf das Eintreten eines Ereignisses zu warten. Zu diesem Zweck bietet SRCP das Kommando `WAIT FB` an. In *rcsh* ist dieser Mechanismus einfach über die Klassen `feedbackXX` erreichbar, wobei `XX` für einen konkreten Rückmeldebaustein steht. Unterstützte Rückmeldeeinheiten sind `M6051` (Klasse `feedbackM6051`), `S88` (`feedbackS88`) und `I8255` (`feedbackI8255`). Die

Klassen `feedbackXX` werden über folgenden Konstruktor instanziiert:

```
def __init__(self, portno, srcp=srcp)
```

Wie bei Loks und Weichen kann ein `srcp`-Objekt spezifiziert werden. Weiterhin muss die Portnummer des Rückmeldekanals angegeben werden. Für einen S88-Rückmeldeport mit der Adresse 38 sieht die Konstruktion so aus:

```
fb = feedbackS88(38)
```

Das somit erzeugte Objekt `fb` kann nun verwendet werden, um auf einen Zustand zu warten. Dazu dient die Methode `wait`:

```
def wait( self, state, timeout ):
    """wait until fb goes into given state.
    wait at most timeout seconds"""
```

`wait()` erwartet zwei Parameter. Der erste Parameter gibt den Zustand — also 0 oder 1 — an, auf den gewartet werden soll. Der Zweite Parameter spezifiziert die maximale Zeit in Sekunden, die die Funktion warten soll. Wird während der Zeitspanne der gewünschte Zustand eingenommen, so gibt die Funktion den Zustand zurück. Sollte die Zeitspanne, die mit `timeout` angegeben wurde, überschritten werden, so gibt die Methode `None` zurück [möglicherweise sollte besser eine Exception generiert werden.] Ein vollständiges Beispiel verdeutlicht die Arbeitsweise:

```
#!/usr/bin/python

from rcsh import *

# der Folgende Code initialisiert einen FB-Signalgeber und
# startet eine Lok. Bei Erreichen des FB stoppt die Lok,
# sonst gibt es einen Sady (heißen die so?)

Connect()
Power()

f = feedbackM6051(6)
l = locoN3(3)
l.setSpeed(4)
l.send()
r = f.wait(1, 40)
if r==1:
    l.setSpeed(0)
    l.send()
if r==0:
    print "hat nich geklappt :=(( "
```

Das kleine Programm definiert ein Rückmeldeport vom Typ M6051, Adresse 6. Außerdem wird eine Lokomotive definiert (N3, Adresse 3). Die Lok wird mit Geschwindigkeit 4 gestartet. Nun wird darauf gewartet, dass die Lok den Zielort erreicht und den Rückmeldeport aktiviert. Bis zu diesem Ereignis wartet das Programm — maximal jedoch 40 Sekunden. Hat die Lok ihr Ziel erreicht, so wird die Geschwindigkeit auf 0 gesetzt, die Lok stoppt.

Die Klasse `feedbackXX` kennt eine weitere Methode: `getState()`. Mit ihrer Hilfe lässt sich der Zustand des aktuellen `feedback`-Ports ermittelt. Ein `rcsh`-Programm kann somit beim Eintreffen eines Ereignisses den Zustand weiterer `fb`-Ports ermitteln, um einen Gesamtzustand des Anlage zu erhalten und entsprechend reagieren.

6. Steuerung durch Rückmelder (asynchron)

Wie synchron auf das Eintreten eines Ereignisses auf einem Rückmeldeport gewartet wird, wurde im letzten Abschnitt vorgestellt. Die Vorgehensweise hat den großen Nachteil, dass das Programm während der Warteperiode keine anderen Aktionen ausführen kann. Erst nachdem das Ereignis eingetreten ist, wird mit dem nächsten Befehl fortgefahren, der dem `wait()` folgt.

Dieser Nachteil kann umgangen werden, indem man die in SRCP vorgesehene Benachrichtigung über FB-Ereignisse ausnutzt. Dazu ist zunächst der `Monitor()` zu starten (s.o.). Es ist unbedingt der `pollport` anzugeben, um den Monitor auf dem Rückmeldepollport lauschen zu lassen. Das `srcp`-Objekt wird somit über FB-Ereignisse benachrichtigt. Es bleibt, eine eigene Logik anzugeben, die beim Eintreffen eines speziellen Ereignisses aufgerufen werden soll. Dazu dienen folgende Methoden der Klasse `srcp_connection` und somit des Objekts `srcp`:

```
def addS88Proc(self, portno, proc):
    self.__addFBProc("S88", portno, proc)

def addI8255Proc(self, portno, proc):
    self.__addFBProc("I8255", portno, proc)

def addM6051Proc(self, portno, proc):
    self.__addFBProc("M6051", portno, proc)
```

Alle Methoden haben gemeinsam, dass sie eine Portnummer und eine Python-Funktion (Callback) erwarten. Die Portnummer sollte selbsterklärend sein. Für die Callback-Funktion gilt: Sie wird aufgerufen, sobald neue Informationen für den spezifizierten Port vorliegen. Als Parameter an die Callback werden die Portnummer und der neue Zustand des sich geänderten Ports übergeben. Dadurch ist es möglich, eine Callback auch auf Änderungen mehrerer FB-Ports reagieren zu lassen. Ein Beispiel zeigt auch hier wieder mehr als tausend Worte:

```
import sys
from rcsh import *

def fbproc(portno, state):
    """will be called (asynchronously) whenever a special feedback
    event occurs"""

    print "FEEDBACK %d: %d" % (portno, state)
    fb4 = feedbackS88(4)
    print "State of FB4: ", fb4.getState()

Connect("localhost", 12345)
# register fbproc as the function to be called whenever the state of
# S88 ports 5 or 6 changes.
srcp.addS88Proc(5, fbproc)
srcp.addS88Proc(6, fbproc)
srcp.monitor(12347, 12346)
# now some other work here...

print "Press Enter to terminate program..."
sys.stdin.readline()
```

Obiges Programm definiert zunächst die Callback-Funktion. Sie hat lediglich die Aufgabe, den neuen Zustand eines sich geänderten FB-Ports auszugeben und anschließend den aktuellen Zustand des FB-Ports 4 zu ermitteln und ebenfalls auszugeben.

Die Callback-Funktion wird für die beiden FB-Ports 5 und 6 installiert. Änderungen auf

einem der beiden Ports werden also an die Callback gemeldet. Der Mechanismus wird mit der Methode `monitor()` gestartet. Dadurch wird ein zweiter Thread gestartet, der nun neben dem Hauptthread läuft. Der Hauptthread (das eigentliche Programm) läuft nun weiter. Würden keine weiteren Befehle im Programm folgen, so würde das Programm terminieren und es würden keine Ausgaben zu beobachten sein. In obigem Programm wartet der Hauptthread daher, bis der Benutzer Enter drückt und terminiert erst dann. Die eigentliche Arbeit wird so lange der zweite Thread machen: Er benachrichtigt `fbproc` über eintretende Ereignisse.

Auf die Verwendung der Funktion `Monitor()`, und somit die Verwendung eines separaten Threads kann verzichtet werden, wenn wie im obigen Beispiel ausschließlich auf Feedback-Ereignisse gewartet und reagiert werden soll, ansonsten jedoch keine Verarbeitung parallel statt finden soll. Dazu muss einfach die Methode `mainloop()` aufgerufen werden wie im folgenden Beispiel:

```
import sys
from rcsh import *

def fbproc(portno, state):
    """will be called (asynchronously) whenever a special feedback
    event occurs"""

    print "FEEDBACK %d: %d" % (portno, state)
    fb4 = feedbackS88(4)
    print "State of FB4: ", fb4.getState()

Connect("localhost", 12345)
# register fbproc as the function to be called whenever the state of
# S88 ports 5 or 6 changes.
srcp.addS88Proc(5, fbproc)
srcp.addS88Proc(6, fbproc)

# now process incoming events in an endless loop
srcp.mainloop()
```

Obiges Programm ist demnach bestens geeignet, eine Steuerung der Eisenbahn zu realisieren, die rein ereignisgesteuert ist. Ein etwas erweitertes Programm zeigt folgendes Listing, das zwei Loks (`l_3` und `l_15`, deren Initialisierung nicht abgedruckt ist) startet und sie stoppt, sobald sie die S88-Rückmeldestellen mit den Nummern 3 bzw. 15 erreichen.

```
def stop15(port, state):
    if not state: return
    l_15.setSpeed(0)
    l_15.send()

def stop3(port, state):
    if not state: return
    l_3.setSpeed(0)
    l_3.send()

srcp.addS88Proc( 3, stop3)
srcp.addS88Proc(15, stop15)

l_15.setSpeed(12)
l_15.send()

l_3.setSpeed(8)
l_3.send()

srcp.mainloop()
```

Generell muss bei der Verwendung der asynchronen Routinen beachtet werden, dass die Zeitdauer, die zur Abarbeitung einer Callback-Funktion benötigt wird, nicht zu lang wird (das gilt natürlich auch für die vorherigen Beispiele). Sonst kann es passieren, dass zu viel Zeit vergeht, bis auf ein einkommendes Ereignis reagiert wird: So lange das Programm mit der Bearbeitung eines Ereignisses beschäftigt ist, werden keine weiteren Ereignisse bedient. Dies kann problematisch werden, wenn neben der ereignisgesteuerten eine zeitgesteuerte Kontrolle der Eisenbahn gemacht werden soll. Die Zeitsteuerung muss wie oben vorgestellt mit Hilfe der Funktion `sleep()` realisiert werden, die jedoch den weiteren Ablauf des Programms verzögert. Um auch eine derartig komplexe Steuerung der Bahn realisieren zu können, müsste auf die Verwendung spezieller Threads zurückgegriffen werden. Dazu müssen lang laufende Callbacks in separaten Threads gestartet werden. Es ist dabei jedoch zu beachten, dass Threads recht viele Ressourcen benötigen und der Start eines Threads relativ lange dauert. Es ist daher wohl etwas experimentieren angesagt, um diesen Ansatz umzusetzen...

Weiterführende Informationen

Die *rcsh* hat mit (oder muss ich sagen: trotz ;-) dieser Dokumentation eine gewisse Verbreitung gefunden. Es haben sich daher bereits einige Skripte ergeben, die von ihren Autoren zur Verfügung gestellt wurden. Diese Skripte befinden sich im Verzeichnis `contrib` der *rcsh*-Distribution. Diese Skripte zeigen Verfahrensweisen auf, wie gewisse Aufgabenstellungen mit Hilfe der *rcsh* gelöst werden können. Sie zeigen Aspekte auf, die in dieser Dokumentation vielleicht etwas zu kurz gekommen sind und sind (hoffentlich) hilfreich bei der Realisierung eigener Projekte.

Sollten nach dieser Dokumentation und obiger Skripte Fragen offen bleiben, so möchte ich noch einmal ermuntern, diese an mich oder an die Autoren der Skripte zu stellen.

... und nun wünsche ich viel Spaß und viel Erfolg mit *rcsh*!

Appendix A – GNU General Public License

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place – Suite 330, Boston, MA 02111–1307, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves,

then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically

receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED

OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS