Ausführliche Netzwerkspezifikation für ECoS2 4.2.10

Communityversion – kein offizielles Dokument

Inhaltsverzeichnis

1 Generelle Überlegungen

1.1 Allgemein

Dieses Dokument richtet sich an (Hobby)Entwickler, die gerne ihre eigene Modellbahnsteuerung über die PC Schnittstelle der ECoS 50210 realisieren möchten (oder einfach mal die Möglichkeiten dafür testen wollen).

Die Inhalte stammen aus der Dokumentation (übersetzt aus dem Englischen), welche über die PC Schnittstelle ausgelesen werden kann (siehe Non Request Befehle) und aus der Dokumentation von 2011 (teilweise veraltet), welche auf der ESU Webseite für registrierte Besitzer einer ECoS verfügbar ist.

Zitat ESU:

«As a general advice, if something seems ambiguous, just give it a try. Most things will be obvious afterward. Please visit our website www.esu.eu»

Die ECoS Zentrale reagiert nicht auf inkorrekte Befehle oder gibt einen Fehlercode zurück.

1.2 Verbotene Nutzung

Es ist ausdrücklich verboten, das Protokoll in einem Server zu implementieren. Insbesondere fällt hierunter die Verwendung des Protokolls in einem zur Steuerung von Modellspielwaren geeigneten Gerät. Dem Lizenznehmer ist es zudem untersagt, auf Grundlage des Protokolls weitere Kommunikationsverfahren zu entwickeln. Dies umfasst ausdrücklich auch die Modifikation des bestehenden Protokolls. Zudem untersagt ist die Implementierung des Protokolls in Software, die auf anderen als PC-Systemen lauffähig ist.
Diese Informationen dürfen nicht Dritten zur Verfügung gestellt werden und stehen unter Lizenz.

Allerdings, Zitat ESU Support (ESU-Forum):

«Solange Sie nur den Client-Teil der Software implementieren und nicht selbst versuchen, ECoS zu spielen (also einem anderen Client vorgaukeln, eine ECoS zu sein), ist aus unserer Sicht alles in Ordnung. Wir können Ihnen nicht verbieten, Ihre Arbeit frei zugänglich zu machen und warum sollten wir das tun?
Ich glaube, Sie haben sich hier in juristischen Fragestellungen gedanklich "verheddert". Es gibt diverse Open-Source Projekte, die erfolgreich eine Kommunikation mit einer ECoS aufbauen und die dann den Code öffentlich zur Verfügung stellen, warum sollte es bei Ihnen anders sein?»

2 Grundlagen

2.1 Mit der ECoS verbinden

Stelle sicher, dass die ECoS entweder mit dem Router oder direkt mit dem PC verbunden ist, wie in der offiziellen Netzwerkspezifikation beschrieben, dort steht auch, wie man die IP Adresse der Zentrale auslesen kann.

Die Zentrale ist über den Port 15471 erreichbar und fungiert als Server, der PC als Client.

2.1.1 Verbinden mit C#

Grundsätzlich gibt es viele Varianten unter unterschiedlichen Programmiersprachen, wie man sich mit der ECoS verbinden kann, ich stelle hier eine Verbindung mit dem TCP-Protokoll unter C# vor, weil ich damit vertraut bin.

Für dieses Protokoll gibt es eine praktische Klasse, nämlich den TCPClient.
Die Verbindung wird standardgemäss aufgebaut:

```
Using System.Net.Sockets

TCPClient client = new TCPClient();
await client.ConnectAsync(ipAdresse, 15471);
```

2.2 Einleitung

Die ECoS hat ein Modell, das ähnlich wie eine Datenbank funktioniert. Die Einheiten im Modell werden Objekte (objects). Jedes Objekt hat eine eindeutige 16-bit object id. Objekte haben Attribute, die den Zustand des Objektes beschreiben. Dabei gibt es auch Array-Attribute, welche mehrere Werte enthalten, wie z.B. Lokomotiven (loco). Bei Array-Attributen wird ein key index benötigt, um auf die einzelnen Werte zuzugreifen. Jedes Objekt gehört zu einer Objektklasse (objectclass), welche die möglichen Attribute festlegt. Ein Objekt kann nach seiner Erstellung seine Klasse nicht mehr wechseln.

Um das Verhalten der ECoS und das Layout zu beeinflussen, müssen die Attribute der Objekte geändert werden.

Teilnehmer (Clients/Participants), welche mit der ECoS verbunden sind, können eine Ansicht (view) von einem Objekt anfordern. Wenn ein Attribut dieses Objekts geändert wird, erhalten alle Beobachter mit einer view einen update event von der ECoS zugeschickt mit dem neuen Wert des Attributes.

Um ein Objekt gegen Änderungen von anderen Clients abzusichern, kann ein Client die Kontrolle (control) erlangen. Kontrollen sind exklusiv, d.h. es kann nur ein Teilnehmer die Kontrolle über ein Objekt haben. Die Kontrolle kann durch das model (die ECoS?) wieder aufgehoben werden. Teilnehmer erhalten keine view Updates bei Objekten, die sie selber kontrollieren und bei Änderungen, die sie selber gemacht haben.

It is implementation specific, if you get view update events for doing changes without having a control. (?)

Mach keine Annahmen darüber, wie die ECoS Anfragen bearbeitet. Das Modell (model) mit den Objekten (objects), Attributen (attributes), Ansichten (views), Kontrollen (controls) und Update Events funktioniert mit dieser Spezifikation (wie man mit der ECoS kommuniziert) und sonst nicht. Genau wie das Modell die Viewers mit Update Events über Änderungen verständigt, verändert es

auch das Layout gemäss den Änderungen im Modell. Wie genau das funktioniert, ist komplett in der ECoS implementiert.

2.3 Netzwerkschnittstelle

Die ECoS nimmt Verbindungen über den Port 15471 entgegen, es existiert ein Limit für die Anzahl der Verbindungen. Die Verbindungen sind komplett isoliert voneinander, abgesehen davon, dass sie die gleichen Daten bearbeiten (->Datenbankmodell).
Die ECoS fungiert dabei als Server und die PC und Smartphones als Clients.

2.4 Werte für Attribute

Attribute und Wahrheitswerte (Boolean) können ohne Parameter(Werte) gesetzt werden. Um sie zu reseten müssen sie auf den Wert 0 gesetzt werden. 0 bedeutet falsch(false), deaktiviert(disable) oder ausgeschaltet(off). 1 bedeutet wahr(true), aktiviert(enable) oder eingeschaltet(on). Aufgepasst, einige Funktionen haben eine invertierte Bedeutung!
Der Standardwert bei Attributen und Booleans ist 0.
Lokale Attribute sind nicht an das globale Modell(model) gebunden, sondern an die PC Schnittstellenverbindung.

2.5 Grundlegende Protokollbeschreibung

2.5.1 Schreibweise

Das Kommunikationsprotokoll ist ASCII und Linienorientiert und achtet auf Gross/Kleinschreibung.
Alle Protokollelemente ausser Texten (strings) sind ASCII7. ASCII32 ist der einzige gültige Leerschlag(Whitespace)charakter.
All ASCII characters less than 32 are control characters, there are no other control characters. (?)

ECoS beendet eine Linie mit CR NL, akzeptiert aber auch NL CR oder CR, dabei sind keine weiteren Charaktere (Buchstaben) erlaubt. Leere Linien oder Linien, welche nur aus Leerschlägen bestehen, müssen ignoriert werden. Elemente dürfen von null oder mehreren Leerschlägen getrennt werden. There could be leading or trailing whitespaces on a line. (?)

Texte(Strings) werden begrenzt von doppelten Anführungszeichen ("). Texte werden in UTF-8 geschrieben.

Control characters and double quotes have to be escaped. Double quotes could be escaped by double quotes (""). Every character can be escaped by double quote, unicodevalue in decimal without leading zeros, double quote, eg. ("10") for NL. No additional whitespaces inside escaping are allowed. If a string terminating double quote is followed by a double quote or a decimal digit, a whitespace must be inserted. Whitespaces inside strings have to be preserved.(?)

Integer(Ganzahlen) werden als Dezimalzahlen geschrieben, auch wenn ECoS 0xh..h als hexadezimal und Oo..o als oktalzahl akzeptiert (Bei Dezimalzahlen dürfen keine 0 vorausgehen).

Nicht jedes Protokollelement, Objekt oder Attribut muss auf der ECoS implementiert werden, aber es wird korrekte Fehlermeldungen senden. Unbekannte oder falsche Daten sollten ignoriert werden und wenn möglich sollte die Kommunikation auf der nächsten Linie fortgesetzt werden.

2.5.2 Pakettypen

Es werden grundsätzlich 3 verschiedene Pakettypen verwendet:
Anfragen (requests), Antworten (replies) und Ereignisse (events).
Zusätzlich dazu gibt es Kommentare (mit # markiert) und leere Linien, welche beide ignoriert werden sollten. Ausserdem gibt es Hilfe (help) und Test (test) Pakete, welche grösstenteils aus Kommentaren bestehen. Kommentare sollten generell nicht in der Releaseversion sein (ausser bei Hilfe und Test).

Der PC sendet eine Anfrage an die ECoS und die ECoS sendet daraufhin eine Antwort.
Antworten werden nur als Reaktion zu einer Anfrage gesendet.
Die ECoS kann view Update Events selbständig an den PC senden. Events werden nicht sortiert zu anderen Events anderer Objekte, aber sie werden sortiert zu Events des gleichen Objekts.
Events und Antworten werden vom PC nicht beantwortet.

Die Paketgrösse ist limitiert, sollte aber für alle normalen Operation genügen, aber gewisse Anfragen können sehr grosse Antworten generieren, welche in einem Grössefehler (size error) enden.

3 Syntax

In der folgenden Dokumentation sind Leerschläge nur zur besseren Darstellung gedacht, es kann mit keinen oder mehreren Leerschlägen gearbeitet werden.

Code	Beschreibung
NL	Neue Linie (ASCII10)
CR	Carriage return (ASCII13) (?)
<[> <]> <(> <)> <"> <-> <_> <,>	Der entsprechende Charakter als Literal
[<text>]	<text> ist optional.
[<text>]+	<text> kann einmal oder mehrmals vorkommen (Aufzählung)
[<text>]*	<text> kann keinmal, einmal oder mehrmals vorkommen.
(<text>)	Gruppe von <text>
<text a> […] <text n>	<text a> oder <text n>, es muss genau einer davon vorkommen. To avoid ambiguity grouping with '(' and ')' will be used.
<ws>	Ein oder mehrere Leerschläge (ASCII32).
<oid>	Objektid als 16 bit Integer
<cmd>	Befehl (command), siehe Befehle. Alphanumerisch mit <_> und <->.
<option>	Option, siehe Optionen. Alphanumerisch mit <_> und <->.
<value>	Kann ein Integer, Text(string), keyword oder ein Parameter der Option sein. In einem Value darf

	kein <,> , <">, <[> oder <]> sein, ausser wenn der Wert nur aus Text besteht.
<errno>	Fehlernummer (error number) als Integer.
<errmsg>	Fehlernachricht (error message), welche nicht als Text sondern als ASCII7 Charaktere besteht. Enthält kein <(> oder <)>.
<commenttext>	Kommentar als Text, sollte vom Parser ignoriert werden. Dies ist kein reiner Text (string), kann aber in UTF-8 sein.
<le>	Linienende.
<parameter>	<[> [<value> [, <value>]*] <]>, die Reihenfolge der Werte ist relevant.
<argument> **<option> [<parameter>]** **<argumentlist>** **<argument> [, <argument>]***	In einer Argumentenliste werden bis zu 10 Argumente unterstützt. Die Reihenfolge der Argumente ist nicht relevant.
<listentry> **<oid> [<ws> <argument>]* CR NL**	Mehrere Argumente werden nur für den Befehl queryObjects benötigt.
<comment> **# <commenttext> <le>**	Kommentar, der ignoriert wird. Darf nicht innerhalb eines Paketes sein. Die ECoS kann Kommentare senden, welche mit CR CL enden.
<help> **help <(> [<helpspec>] <)> <le>**	Generiert ein Hilfspaket, welches Infos über die Verwendung dieser Schnittstelle gibt.
<test> **test <(> <teststr> [, nocrnl] <)> <le>**	<teststr> wird an die ECoS gesendet, welche ihn daraufhin zurücksendet. The string in <teststr> will be replied unquoted and unescaped. Without the option nocrnl the command station will add a trailing CR NL. It's possible for a command station to accept tests without trailing <le>, but this is not recommended. The test packet is not a request, as it neither fullfills the syntactical requirements, nor does it normally generate a reply, although it can generate a reply on error.
<request> **<cmd> <(> <oid> [, <argumentlist>] <)>** **<le>**	Anfrage (request), welche an die ECoS gesendet wird. Diese wird mit einer Antwort (reply) reagieren. Die Zentrale akzeptiert Anfragen ohne abschliessendes <le>, dies ist jedoch nicht empfohlen. Obwohl praktisch alle Anfragen <oid> zwingend benötigen, sind formal alle Argumente Teil einer optionalen Argumentenliste, so dass theoretisch keine Argumente übergeben gesendet werden könnten.
<reply> **< REPLY <ws> (<cmd> <(> <oid> [,** **<argumentlist>] <)>) | <?> > CR**	Antwort (reply) der ECoS als Reaktion auf eine Anfrage (request). The request repetition is at the moment a plaintext copy of the request, stripped of

NL [<listentry>]* < END <ws> <errno> <ws> <(> <errmsg> <)> > CR NL	leading and trailing whitespaces. It has yet to be decided, if the request repetition could be reformated and reordered. In this case the request repetition will not have more errors than the request and the request repetition will not be semantically changed. Formally the arguments for the command in the request repetition are an optional <argumentlist>, see also above in request. The command station can replace the request repetition by a single <?>, if it can't generate a valid request repetition. This must result in an error, eg. errno != 0. The order of listentrys is arbitrary.
<event> **< EVENT <ws> <oid> > CR NL [<listentry>]+ < END <ws> <errno> <ws>** **<(> <errmsg> <)> > CR NL**	Events werden von der ECoS generiert und selbständig gesendet, wenn das dazugehörige Objekt mit einer view abonniert ist. Die Reihenfolge der listentry ist willkürlich.
Alles andere	Linien, welche nur Leerschläge (ASCII32) und Linien, welche mit # starten (Kommentare) werden von der Zentrale ignoriert und lösen keine Antwort (reply) aus.

4 Befehle

4.1 Non Request Befehle

Diese Befehle (mit Ausnahme von test()) geben Auskunft über die Funktionsweise der PC-Schnittstelle zur ECoS.
Diese Auskunft Nachrichten enden geben nie das Ende der Nachricht an. Jede Zeile beginnt mit einem # (Bezeichnet die Zeile als Kommentar), deshalb zählen diese Befehle nicht als Request.

Befehl	Beschreibung
help()	Gibt eine Antwort mit der Übersicht aller Hilfe-Befehle zurück.
help(intro)	Gibt als Antwort einen Einstieg in die ECoSNet Schnittstelle.
help(basic)	Gibt als Antwort eine grundlegende Beschreibung des verwendeten Protokolls der ECoSNet.
help(syntax)	Gibt als Antwort eine Beschreibung des Syntax der ECoSNet.
help(command)	Gibt als Antwort eine kurze Erklärung der verfügbaren Befehle (ohne die Hilfe-Befehle).
help(error)	Gibt als Antwort eine Auflistung der implementierten Fehlercodes.

help(<object>, [<command>, [<option>]])	Gibt als Antwort einen genaueren Beschrieb zu diesem Objekt.
test(<teststring>,[nocrnl])	The command station will reply the supplied teststring unquoted and unescaped. Without the option nocrnl the command station will add a trailing CR NL. The test packet is not a request, as it neither fullfills the syntactical requirements, nor does it normally generate a reply, although it can generate a reply on error. Shouldn't be used in a release version.

Beispiel zu help(<object>, [<command>, [<option>]]):

Als <object> können alle Objekte aus der objectclasses verwendet werden,
möglicher Befehl; help(model).

4.2 Request Befehle

Auf Request-Befehle reagiert die ECoS immer mit einer Antwort, allenfalls mit Fehlercode.
Dabei wird der Anfang und das Ende der Antwort immer klar markiert. Im Header der Antwort steht zudem immer der an die ECoS gesandte Befehl.

Befehl	Beschreibung	Beispiele
set	Setzt den Wert eines Attributes eines Objekts. Manchmal braucht man die Kontrolle über das jeweilige Objekt. Man kann mit einem Befehl mehrere Parameter setzel. Attribute zählen so als Optionen mit ihren Werten als Parameter. Es ist möglich, die derzeitigen Werte zu erhalten, wenn ein Fehler sich ereignet.	Lokomotive(Geschwindigkeit): set(1000, speed[120]) Arrayattribut setzen; Lokomotive(Licht einschalten) set(1000, func[0, 1]
get	Fragt den Wert eines Attributes eines Objekts ab. Es wird empfohlen, diese mit den view Updates zu tun. Attribute sind als Optionen deklariert, ihre Werte als Parameter. Gewisse Metaattribute sind nur lesbar, haben aber keinen eigenen Wert, sie geben nur das Attribut zurück ohne Wert. Array Attribute sind als Option deklariert, wobei der Index der erste Parameter ist und der Wert der zweite. Wenn du nicht spezifische Optionen angibst, werden sämtliche Werte der Attribute zurückgegeben, was eine riesige Liste sein kann. Wenn die Liste zu lang wird, werden Einträge durch die Zentrale mit <attribute> <[> ... <]> listentry ausgeblendet, dies generiert keinen Fehler.	
create	Erstellt ein neues Objekt. Ein neu erstelltes Objekt ist privat, bis es mit append public	

	gemacht wird. Es kann nur immer ein privates Objekt existieren.	
delete	Löscht ein Objekt, manchmal wird die Kontrolle über das zu Löschende Objekt benötigt.	
request	Die Kontrolle oder eine View über ein Objekt wird angefragt.	
release	Hebt die Kontrolle oder View über ein Objekt wieder auf.	
link	Fügt ein Objekt einem ManagerObjekt hinzu.	
unlink	Entfernt ein Objekt von einem Manger Objekt.	
queryObjects	Fragt eine Liste von Objekten ab, welche von diesem Objekt verwaltet werden.	

5 Objektklassen

Objektklassen stellen einerseits die Eigenschaften und Zustände der ECoS (wie z.B. Stop & Go Taste), angeschlossene Geräte (wie z.B. Switchpilots, ECoSDetectors und Booster) und natürlich Loks dar und erlauben den Zugriff und das Bearbeiten dieser Geräte.

5.1 Model

5.1.1 Spezifikation

Bezeichnung	Wert
Objectclass	model
ID	1 (baseobject)
Manager	-
Manages	Alle baseobjects

5.1.2 Beschreibung

Das Model ist die Basis für alle Objekte im ECoSNet Protokoll. Du kannst hier einige Konfigurationen an der ECoS vornehmen, Stop und Go werden hier verwaltet, sowie m4 Status, RailComPlus Registrierung und das Programmgleis.

5.1.3 Attribute

Attribut	Beschreibung	Mögliche Befehle
objectclass		
listview		
view		
control		
list		

size		
minarguments		
protocolversion		
commandstationtype		
name		
serialnumber		
hardwareversion		
applicationversion		
applicationversionsuffix		
updateonerror		
status (Invertierter Status über STOP / GO / SHUTDOWN)		
status2		
prog-status		
m4-status		
railcomplus-status		
watchdog		
railcom		
railcomplus		
railcomplus-range		
railcomplus-mode		
allowlocotakeover		
stoponlastdisconnect		

5.2 Programmiergleis

5.2.1 Spezifikation

Bezeichnung	Wert
Objectclass	programmingtrack
ID	5
Manager	1 (model)
Manages	-

5.2.2 Beschreibung

Das Programmiergleisobjekt gibt Zugang zum Programmierservice der ECoS. Jeder Programmierbefehl erstellt einen Programmierjob, die Antwort der Zentrale gibt nur Auskunft über den Erfolg des Erstellens. Sobald die Programmierung abgeschlossen ist, erhältst du einen Update Event mit dem Resultat, demzufolge brauchst du eine view zum Programmiergleis. Mehrere Programmiersjobs können in der Todoliste stehen, sie werden einer nach dem anderen abgearbeitet.

5.2.3 Attribute

Attribut	Beschreibung	Mögliche Befehle
objectclass		
view		
listview		
control		
programmingjob		

5.3 Hauptgleis

5.3.1 Spezifikation

Bezeichnung	Wert
Objectclass	pom
ID	7 (baseobject)
Manager	1 (model)
Manages	-

5.3.2 Beschreibung

Das pom Objekt gibt Zugang zum Programmieren auf dem Hauptgleis der ECoS. Jeder Befehl erstellt einen Programmierjob, wenn mehrere vorhanden sind, werden sie einer nach dem Andern abgearbeitet. Die Antwort bestätigt nur den Erfolg des Erstellens, nach dem Beenden der Programmierung sendet die Zentrale einen Update Event mit dem Status an die Viewer.

5.3.3 Attribute

Attribut	Beschreibung	Mögliche Befehle
objectclass		
view		
listview		
control		
programmingjob		

5.4 Lokmanager

5.4.1 Spezifikation

Bezeichnung	Wert
Objectclass	loco-manager
ID	10
Manager	1 (model)
Manages	Alle Loks und Multitraktionen

5.4.2 Beschreibung

Der Lokmanager verwaltet alle erfassten Loks und Multitraktionen der ECoS.

5.4.3 Attribute

Attribut	Beschreibung	Mögliche Befehle
objectclass		
view		
listview		
control		
list		
size		
protocol		
id		
autoregister		
addr		
locodesc		
name		
favorite		
active		

5.5 Lok

5.5.1 Spezifikation

Bezeichnung	Wert
Objectclass	loco
ID	Dynamic (startet von 1000)
Manager	10 (loco-manager)
Manages	-

5.5.2 Beschreibung

Stellt eine normale Lokomotive/Fahrzeug dar.

5.5.3 Attribute

Attribut	Beschreibung	Mögliche Befehle
objectclass		
view		
listview		
control		
autoregister		
protocol		
addr		
sniffer		
locodesc		
symbol		
name		
favorite		
active		
addressconflict		

dir		
speed		
speedstep		
speedindicator		
func		
funcset		
funcicon		
funcsymbol		
funcsymbolex		
funcdesc		
funcexists		
multi		
profile		
cv		

5.6 Lok-Multitraktion

5.6.1 Spezifikation

Bezeichnung	Wert
Objectclass	loco-multi
ID	Dynamic (startend ab 1000)
Manager	10 (loco-manager)
Manages	-

5.6.2 Beschreibung

Stellt eine Multitraktion aus mehreren Loks/Fahrzeugen dar.

5.6.3 Attribute

Attribut	Beschreibung	Mögliche Befehle
objectclass		
view		
listview		
control		
list		
size		
autoregister		
protocol		
addr		
sniffer		
locodesc		
symbol		
name		
favorite		
active		
addressconflict		
dir		

speed		
speedstep		
speedindicator		
func		
funcset		
funcicon		
funcsymbol		
funcsymbolex		
funcdesc		
funcexists		
multi		
reverse		

5.7 Schaltartikel-Manager

5.7.1 Spezifikation

Bezeichnung	Wert
Objectclass	accessory-manager
ID	11 (baseobject)
Manager	1 (model)
Manages	Alle Magnetartikel, Weichen etc., Fahrstrassen und Drehscheiben

5.7.2 Beschreibung

Der Schaltartikelmanager verwaltet alle schaltbaren Schalt-/Magnetartikel (Weichen, etc.), Drehscheiben und Fahrstrassen.

5.7.3 Attribute

Attribut	Beschreibung	Mögliche Befehle
objectclass		
view		
listview		
control		
list		
size		
protocol		
switch		
id		
type		
mode		
addr		
addrext		
symbol		
name1		
name2		
name3		

5.8 Schaltartikel

5.8.1 Spezifikation

Bezeichnung	Wert
Objectclass	accessory
ID	Dynamic (startend von
Manager	11 (accessory manager)
Manages	-

5.8.2 Beschreibung

Ein Magnet- / Schaltartikel repräsentiert eine Weiche, Signal, Entkupplungsgleis oder Ähnliches. Diese werden z.B. mit einem SwitchPilot geschalten.

5.8.3 Attribute

Attribut	Beschreibung	Mögliche Befehle
objectclass		
view		
listview		
control		
state		
type		
protocol		
addr		
addrext		
mode		
duration		
symbol		
gates		
variant		
name1		
name2		
name3		
position		
switching		

5.9 Fahrstrasse

5.9.1 Spezifikation

Bezeichnung	Wert
Objectclass	Route
ID	Dynamic (startend von
Manager	11 (accessory manager)
Manages	Alle vorhandenen Schaltartikel in dieser Fahrstrasse

5.9.2 Beschreibung

Eine Fahrstrasse ist ein Makro zum Schalten von mehreren Schaltartikeln. Sie kann aufgrund von benutzerdefinierten Bedingungen (z.B. eine Belegung eines bestimmten Gleisabschnittes) auch automatisch von der ECoS eingestellt werden.

5.9.3 Attribute

Attribut	Beschreibung	Mögliche Befehle
objectclass		
view		
listview		
control		
list		
size		
state		
type		
protocol		
addr		
addrext		
mode		
duration		
symbol		
delay		
timing		
retrigger		
group		
name1		
name2		
name3		
switching		
state		

5.10 Drehscheibe

5.10.1 Spezifikation

Bezeichnung	Wert
Objectclass	turntable
ID	Dynamic (startend von
Manager	11 (accessory manager)
Manages	-

5.10.2 Beschreibung

Stellt den Zugriff auf eine Drehscheibe dar.

5.10.3 Attribute

Attribut	Beschreibung	Mögliche Befehle
objectclass		
view		
listview		
control		
state		
type		
protocol		
addr		
addrext		
mode		
duration		
symbol		
name1		
name2		
name3		

5.11 Schnittstellenmanager

5.11.1 Spezifikation

Bezeichnung	Wert
Objectclass	interface-manager
ID	19 (baseobject)
Manager	1, (model)
Manages	Alle interfaces (Schnittstellen), momentan das Wifi-Interface

5.11.2 Beschreibung

Der Schnittstellenmanager verwaltet alle Schnittstellen. Die tatsächliche Implementierung ist ein Prototyp und Gegenstand von Veränderungen.

5.11.3 Attribute

Attribut	Beschreibung	Mögliche Befehle
objectclass		
view		
listview		
control		
list		
size		

5.12Wifi-Schnittstelle

5.12.1 Spezifikation

Bezeichnung	Wer
Objectclass	wifi-interface
ID	Dynamic
Manager	19 (interface-manager)
Manages	-

5.12.2 Beschreibung

Stellt eine Wifi-Schnittstelle dar (wahrscheinlich mit dem Accesspoint zur Mobil Control II).

5.12.3 Attribute

Attribut	Beschreibung	Mögliche Befehle
objectclass		
view		
listview		
vontrol		
wifi-mode		
wifi-channel		
wifi-ssid		

5.13Feedback-Manager

5.13.1 Spezifikation

Bezeichnung	Wert
Objectclass	feedback-manager
ID	26
Manager	1 (model)
Manages	Alle Rückmeldemodule

5.13.2 Beschreibung

Der Feedbackmanager verwaltet alle Rückmeldemodule (Detector, S88, etc.).

5.13.3 Attribute

Attribut	Beschreibung	Mögliche Befehle
objectclass		
view		
listview		
control		

list		
size		
ports		

5.14 Rückmeldemodul

5.14.1 Spezifikation

Bezeichnung	Wert
Objectclass	feedback-module
ID	Dynamic (startend von
Manager	25 (feedback-manager)
Manages	-

5.14.2 Beschreibung

Ein Rückmeldemodul gewährt Zugriff auf einen S88 oder auf einen ECoSDetector.

5.14.3 Attribute

Attribut	Beschreibung	Mögliche Befehle
objectclass		
view		
listview		
control		
ports		
state		
railcom		

5.15 Booster-Manager

5.15.1 Spezifikation

Bezeichnung	Wert
Objectclass	booster-manager
ID	27
Manager	1 (model)
Manages	Alle Booster

5.15.2 Beschreibung

Der Boostermanager verwaltet alle angeschlossenen Strombooster.

5.15.3 Attribute

Attribut	Beschreibung	Mögliche Befehle
objectclass		

view		
listview		
control		
list		
size		

5.16 Booster

5.16.1 Spezifikation

Bezeichnung	Wert
Objectclass	booster
ID	Dynamic (startend von
Manager	27 (booster-manager)
Manages	-

5.16.2 Beschreibung

Erlaubt den Zugriff auf einen Booster der ECoS.

5.16.3 Attribute

Attribut	Beschreibung	Mögliche Befehle
objectclass		
view		
listview		
control		
maxlimit		
limit		

5.17 Stellpult-Manager

5.17.1 Spezifikation

Bezeichnung	Wert
Objectclass	accessorylayout-manager
ID	31
Manager	1 (model)
Manages	-

5.17.2 Beschreibung

Das Stellpult ermöglicht den direkten Zugriff auf das Stellpult der ECoS.

5.17.3 Attribute

Attribut	Beschreibung	Mögliche Befehle

objectclass		
view		
listview		
control		
layout		
link		

5.18Lokbild-Manager

5.18.1 Spezifikation

Bezeichnung	Wert
Objectclass	locoimage-manager
ID	40
Manager	1 (model)
Manages	-

5.18.2 Beschreibung

Der Lokbildmanager verwaltet die Metadaten der Lokbilder der ECoS (vordefinierte und benutzerdefinierte). Trotzdem können die Lokbilder nicht über dieses Objekt abgefragt werden, sondern nur der Bildindex, der Bildname und der Bildtyp. Das effektive Bild kann nur über die Bild-API abgefragt werden.

5.18.3 Attribute

Attribut	Beschreibung	Mögliche Befehle
objectclass		
view		
listview		
control		
type		
name		

6 Fehlercodes

Bei einer Antwort (reply) der ECoS wird normalerweise ein Fehlercode zurückgegeben. 0 Bedeutet in diesem Fall OK und dass die Zentrale den Befehl ausführen konnte. Alle anderen Codes bedeuten einen Fehler und dass die Zentrale die Verarbeitung abgebrochen hat.
Die Zentrale kann trotz Code 0 mit einer Warnung antworten, tut sie dies nicht ist die errormessage OK.

Fehlercode
0 OK
0 OK, but no newline after packet
0 OK, but no attribute
0 OK, control not implemented and not needed

0 OK, but already have control
0 OK, but already free
0 OK, but not controller
0 OK, but no view changed
0 OK, but obsolete option at 0
0 OK, but obsolete attribute at 0
0 OK, but function not exists at 0
0 OK, but function not used at 0
0 OK, but string shortened at 0
0 OK, but wrong fallback at 0
0 OK, but no RailCom port at 0
0 OK, but index not used at 0
10 unknown command at 0
11 unknown option at 0
11 unknown objectclass at 0
11 invalid option at 0
11 size with other arguments
11 append contradicts discard
11 protocol/addr with other attributes
12 invalid parameter at 0
12 no data for switch at 0
12 contradicting parameters at 0
12 unknown parameter at 0
13 invalid character in string at 0
13 NULL in string at 0
15 unknown object at 0
18 internal error
18 internal error at 0
19 invalid character at 0
20 line to long, 0 bytes
20 to many arguments at 0
21 syntax error, "'" expected at 0
21 invalid ucs code at 0
21 syntax error at 0
21 empty parameterlist at 0
21 invalid utf-8 coding at 0
21 leading zero in decimal at 0
21 invalid character in integer at 0
21 integer out of range at 0
21 syntax error, '(' expected at 0
21 syntax error, ')' expected at 0
21 syntax error, ']' expected at 0
22 function not used at 0
22 invalid state at 0
22 read only attribute at 0
22 const attribute at 0
22 attribute can't be used at create at 0
22 no queryable attribute at 0
22 cannot insert at 0
22 cannot remove at 0
22 not in list at 0

22 not allowed on unappended object at 0
22 discard with other options at 0
22 max objects reached at 0
22 not possible at 0
22 not allowed at 0
22 bad wifi mode at 0
22 invalid object at 0
22 not allowed
22 no link possible
22 no unlink possible
22 no managerobject
22 invalid address at 0
22 invalid protocol at 0
22 function not exists at 0
23 duplicate option at 0
25 control needed
25 controlled by somebody else
25 couldn't get control
28 no object to discard at 0
29 no add
29 force without control
29 update without view
29 attribute without view
29 listattribute without view
29 viewswitch without view
29 needs append at 0
29 no id
29 no state
29 no view or control
29 missing addr
29 missing mode
29 missing cv
30 missing funcset-parameter at 0
30 missing addrext-parameter at 0
30 missing switch-parameter at 0
30 missing cv-parameter at 0
30 missing parameter at 0
30 missing bool-parameter at 0
30 missing 0/1-parameter at 0
30 missing integer-parameter at 0
30 missing integer-parameter at 0
30 missing string-parameter at 0
30 missing string-parameter at 0
30 missing string-parameter at 0
30 missing keyword-parameter at 0
30 missing attribute-parameter at 0
30 missing listattribute-parameter at 0
31 not implemented at 0
31 only appending modules implemented at 0
31 no control possible
31 only for last module implemented

32 no RailComPlus without RailCom at 0
32 no go after stop
32 address conflict at 0
35 already created at 0

7 REST Schnittstelle

Die REST Schnittstelle/API ist eigentlich nicht Teil des offiziellen PC-Interfaces der ECoS, aber da sie nützlich ist und kein eigenes Dokument hat, wird sie hier ebenfalls aufgeführt.

Anmerkung vom ESU Support aus dem Forum:

«An sich ist die Schnittstelle nicht Teil der Spezifikation der ECoS, d.h. sie kann sich jederzeit ohne Vorankündigung ändern, insbesondere was das Bildformat (bezüglich der Lokbild-API, Anm. des Autors) betrifft. Da sie aber von der MC2 verwendet wird, ist sie relativ sicher.
Es gibt in der aktuellen Version noch eine andere Möglichkeit (momentan unbekannt, Anm. des Autors). Diese bitte nicht mehr verwenden, da sie in der nächsten Release rausfliegt.»

7.1 Spezifikation

API-Endpoint	Beschreibung	Rückgabewerte
***/loco/image?type=user&index=0**	API zum Abfragen von benutzerdefinierten Lokbildern. Indexierung der Bilder beginnt immer mit 0. Der genaue Index kann mit der PC-Schnittstelle beim Lokbildmanager abgefragt werden.	.bmp Lokbild (ESU Standard) oder Error404, wenn das Bild nicht gefunden wurde
***/loco/image?type=internal&index=0**	API zum Abfragen von systemdefinierten Lokbildern. Indexierung der Bilder beginnt immer mit 0. Der genaue Index kann mit der PC-Schnittstelle beim Lokbildmanager abgefragt werden.	.bmp Lokbild (ESU Standard) oder Error404, wenn das Bild nicht gefunden wurde

* = IP-Adresse der ECoS

8 Releasenotes PC-Interface 2.0

Wir haben die Pufferverwaltung geändert, und kurzfristig kann die PC-Verbindung auf ein Vielfaches an Speicher zugreifen, wie bisher, so dass wir davon ausgehen, das unter normalen Lastszenarien keine Pufferprobleme mehr auftreten.

Der Parser sollte keine Fehler mehr durchgehen lassen. Außerdem sind die Fehlermeldungen nun deutlich aussagekräftiger.

Man kann im Setup 3 aktivieren, das Fehlermeldungen des PC-Interfaces im ECoS-Log eingetragen werden.

Die ECoS kann im Objekt 1 einen Watchdog für die PC-Verbindung aktivieren. Optional kann beim Timeout ein Stop ausgelöst werden. Siehe auch help(1,set,watchdog).

Im Objekt 1 kann man nun die blinkende Go-Taste (status2), M4-Anmeldung (m4-status), RailComPlus-Anmeldung (railcomplus-status) und Programmierstatus (prog-status) abfragen.

Außerdem kann man ins ECoS-Log schreiben (writelog).
Das Schalten-Symbol (die beiden Pfeile bei schaltenden Schaltartikeln) sind nun im PC-Interface.
Man kann beim Erzeugen einer View mit der update Option sich die aktuellen Werte geben lassen, so das man sich das get sparen kann.

Man kann mit der Option updateonerror global aktivieren, das (einige) fehlgeschlagene set Anweisungen automatisch den aktuellen Wert zurückliefern.

Man kann get auch verallgemeinert benutzen, z.B. liefert get(1000,func) alle existierenden Funktionen zurück und get(1000) liefert alle implementierten Attribute zurück.

Bisher haben Views auf Managerobjekte keine Informationen zurückgeliefert, was sich geändert hat. Es wird nun auch mitgeliefert, welches Objekt sich geändert hat.
Bei Managerobjekten kann man sich nun auch mit request(10,view,listattribute[addr]) auch Updates auf Änderungen von Attributen der gemanagten Objekte geben lassen.

Mit der attribute Option kann man die View auf ein Objekt einschränken.
Es gibt nun eine einfache Implementierung (Read-Only, keine Update-Events) des Stellwerkmanagers. Mit get(31,link[0]) kann man sich z.B. die Belegung der ersten Stellwerkseite geben lassen.

Zugriff auf das Programmiergleis der ECoS. Wichtig: Man benötigt eine View auf das Programmiergleis, da die Daten mit einem Update Event kommen. Implementiert sind DCC Direct-Mode lesen/schreiben und Motorola 6021-Mode schreiben. Bsp: request(5,view) set(5,mode[readdccdirect],cv[1],cv[2]). Man Beachte: set statt get, auch beim Lesen. Weitere Infos mit help(5,set,all).

9 Unbearbeitete help() Ausgaben

9.1 Implemented objectclasses

model
programmingtrack
pom
loco-manager
accessory-manager
interface-manager
feedback-manager
booster-manager

accessorylayout-manager
locoimage-manager
loco
loco-multi
accessory
route
turntable
feedback-module
booster
wifi-interface

9.2 Introduction to ECoSNet

The command station has a model, which is kind of a database. The
entities in the model are called objects. Objects are identified by a
16-bit object id. Objects have attributes, which contain the object
state. Array attributes are attributes with more than one value, eg.
function state of a loco. A key or index is needed to select the value
of an array attribute. Every object has a object class, which specifies
its attribute set. An object can't change its class after creation.
#
To change the behavior of the command station and hence the layout,
attributes of objects have to be modified.
#
Participants can have a view on an object. If an attribute of an object
changes, all viewers get an view update event with the new value of the
attribute. This kind of updating is prefered over polling.
#
To lock an object against changes from other participants you can obtain
a control on an object. Controls are exclusive, eg. nobody can have
another control on the same object. Controls can be removed by the
model. Controller don't get view update events for changes they have
done themselves. It is implementation specific, if you get view update
events for doing changes without having a control.
#
Never ever make any assumptions on how the command station handles
things. There is a model with objects, attributes, views, controls and
update events, and this specification on how to communicate with the
model, nothing more. Like the model updates viewers on the pc-interface,
it will change the environment, eg. layout, according to the changes in
the model. How this is done is completely implementation specific by the
command station.
#

9.3 Network specification

TCP 15471, there is a limit of connections. Different connections to the
command station are isolated against each other, except that they work
on the same model. The command station implements a server, pcs and

smartphones are clients.
#

9.4 Special option types

Boolean options and attributes can be set without parameter. To reset
them 0 as parameter has to be used. 0 meaning false, disable or off.
1 meaning true, enable or on. Be aware, that some options have a negated
meaning. Boolean options/attributes defaults to 0 (false).
#
Local attributes are not bound to an global model object, but are bound
to the pc interface connection.

9.5 Basic protocol description

The protocol is ASCII and line oriented. The protocol is case sensitive.
All protocol elements outside strings are ASCII7. ASCII32 is the only
valid whitespace character. All ASCII characters less than 32 are control
characters, there are no other control characters. ECoS2 finishes a line
with CR NL, but also accepts NL CR, NL or CR. No additional control
characters are allowed. Empty lines, eg. lines with only whitespaces,
have to be ignored. Elements can be seperated by zero or more
whitespaces. There could be leading or trailing whitespaces on a line.
#
Strings are limited by double quotes ("). Strings are represented in
UTF-8. Control characters and double quotes have to be escaped.
Double quotes could be escaped by double quotes (""). Every character can
be escaped by double quote, unicodevalue in decimal without leading
zeros, double quote, eg. ("10") for NL. No additional whitespaces inside
escaping are allowed. If a string terminating double quote is followed by
a double quote or a decimal digit, a whitespace must be inserted.
Whitespaces inside strings have to be preserved.
#
Integers are printed in decimal, although ECoS accepts 0xh..h as
hexadecimal and 0o..o as octal (so no leading zeros for decimals).
#
Not every protocol element, object or attribute has to be implemented on
the command station, but it will send proper error messages. Unknown or
wrong data should be ignored, and if possible continue communication on
the next line.
#
There are 3 types of packets: requests, replies and events.
Also there could be comments and empty lines, which should be ignored,
and special test and help packets. Comments shouldn't be in release
versions of software, but the parser still must be able to parse and
ignore them.
The pc sends requests to the ECoS, the ECoS answeres with a reply.
Replies are in order to the requests.
The ECoS can send view update events to the pc. Events must not be in

```
#   order to events from other objects or replies, but they must be in order
#   to events from the same object.
#   Events or replies are never confirmed by the pc.
#   There is a limit on total packet size. This limit is large enough for all
#   normal operation, but be aware, that some requests can generate very
#   large replies, thus resulting in size errors.
```

9.5.1 Syntax

```
#   In the following documentation whitespaces are for clarity only, there
#   can be zero or more whitespaces.
#
#   NL
#     New line (ASCII10).
#   CR
#     Carriage return (ASCII13).
#   <[> <]> <(> <)> <"> <-> <_> <,>
#     The corresponding character as literal.
#   [ <text> ]
#     <text> is optional.
#   [ <text> ]+
#     <text> can be one or multiple times.
#   [ <text> ]*
#     <text> can be zero, one or multiple times.
#   ( <text> )
#     Grouping of <text>.
#   <text a> | ... | <text n>
#     <text a> or ... or <text n>. There must be exact one.
#     To avoid ambiguity grouping with '(' and ')' will be used.
#   <ws>
#     One or more whitespaces (ASCII32).
#   <oid>
#     Object id as 16 bit decimal integer.
#   <cmd>
#     Command, see below. Alphanumeric with <_> and <->
#   <option>
#     Option, see below. Alphanumeric with <_> and <->
#   <value>
#     Can be a integer, a string, a keyword or anything else as
#     described for the parameter of the option. There will be no <,>,
#     <">, <[> or <]> in a value, except the value solely consists of
#     a string.
#   <errno>
#     Error number as integer.
#   <errmsg>
#     Error message (unquoted). Implementation specific. There will be no
#     <(> or <)> in an errmsg. Since this is no string, it must solely
#     consists of ASCII7 characters without control characters and without
#     <(> and <)>.
```

```
#   <commenttext>
#     Comment as text, should be ignored by parser. This is not a string,
#     but can be in UTF-8 without control characters and without escaping.
#   <le>
#     Line end, eg. CR or NL.
#
#   <parameter>
#     <[> [ <value> [ , <value> ]* ] <]>
#     The order of values is relevant.
#   <argument>
#     <option> [ <parameter> ]
#   <argumentlist>
#     <argument> [ , <argument> ]*
#     In an argument list, at least 10 arguments must be supported.
#     The order of the arguments is not relevant
#   <listentry>
#     <oid> [ <ws> <argument> ]* CR NL
#     Multiple arguments are only used for replies of queryObjects
#     commands.
#
#   <comment>
#     # <commenttext> <le>
#     Comment. Must be ignored. Must not be inside a packet.
#     Command station will send comments terminating with CR CL.
#   <help>
#     help <(> [ <helpspec> ] <)> <le>
#     The help packet generates a help answere. All answeres generated by
#     a help packet will be a valid comment. Its possible for a command
#     station to accept help packets without trailing <le>, but this is not
#     recommended. The help packet is not a request, as it neither
#     fullfills the syntactical requirements, nor does it normally generate
#     a reply, although it can generate a reply on error. The syntax of the
#     help packet and the generated answere is completely implementation
#     specific, so consult help() for help about help.
#   <test>
#     test <(> <teststr> [ , nocrnl ] <)> <le>
#     Test string send to command station. The string in <teststr> will be
#     replied unquoted and unescaped. Without the option nocrnl the command
#     station will add a trailing CR NL. It's possible for a command station
#     to accept tests without trailing <le>, but this is not recommended.
#     The test packet is not a request, as it neither fullfills the
#     syntactical requirements, nor does it normally generate a reply,
#     although it can generate a reply on error.
#   <request>
#     <cmd> <(> <oid> [ , <argumentlist> ] <)> <le>
#     Request send to command station. The command station will answere
#     with a reply. Its possible for a command station to accept requests
#     without trailing <le>, but this is not recommended and should be
#     acknowledged with a warning. Although up to now all requests need an
```

```
#       <oid> as a mandatory first argument, formally all arguments of a
#       request are an optional <argumentlist>, eg. there could be no
#       arguments or a non integer first argument.
#    <reply>
#       < REPLY <ws> ( <cmd> <(> <oid> [ , <argumentlist> ] <)> ) | <?> > CR
#       NL [ <listentry> ]* < END <ws> <errno> <ws> <(> <errmsg> <)> > CR NL
#       Reply from the command station to a request. The request repetition
#       is at the moment a plaintext copy of the request, stripped of leading
#       and trailing whitespaces. It has yet to be decided, if the request
#       repetition could be reformated and reordered. In this case the
#       request repetition will not have more errors than the request and
#       the request repetition will not be semantically changed. Formally the
#       arguments for the command in the request repetition are an optional
#       <argumentlist>, see also above in request. The command station can
#       replace the request repetition by a single <?>, if it can't generate
#       a valid request repetition. This must result in an error,
#       eg. errno != 0. The order of listentrys is arbitrary.
#    <event>
#       < EVENT <ws> <oid> > CR NL [ <listentry> ]+ < END <ws> <errno> <ws>
#       <(> <errmsg> <)> > CR NL
#       Events can be generated by the command station as a result of a view.
#       The order of listentrys is arbitrary.
```

9.5.2 Other lines

```
#     Lines only containing whitespaces (ASCII32) (empty lines) and lines
#     starting with '#' (comments) are ignored by the command station and
#     will result in no reply.
```

9.5.3 Commands, which are not requests

```
#    help
#       The help will generates a help answere. All answeres generated by a
#       help command will be a valid comment. The help packet is not a
#       request, as it neither fullfills the syntactical requirements, nor
#       does it normally generate a reply, although it can generate a reply
#       on error. The syntax of the help packet and the generated answere is
#       completely implementation specific, so consult help() for help about
#       help. In contrast to the test packet, which is well defined, the
#       implementation of the help packet can violate anything in the
#       rest of the specification. Shouldn't be used in a release version.
#    test
#       The command station will reply the supplied teststring unquoted and
#       unescaped. Without the option nocrnl the command station will add a
#       trailing CR NL. The test packet is not a request, as it neither
#       fullfills the syntactical requirements, nor does it normally generate
#       a reply, although it can generate a reply on error. Shouldn't be used
#       in a release version.
```

#

9.5.4 Commands, which are requests

```
#   set
#     Set the value of an attribute in the object. You sometimes need a
#     control to do this. You can set more than one value with one set
#     command. Attributes are specified as option with their value as
#     parameter.
#     Example: set(1000, speed[120]) for setting the speed of a loco.
#     Array attributes need an index, which value should be set.
#     Example: set(1000, func[0, 1] for putting on the light of a loco on
#     F0/FL.
#     It is possible to get returned the actual values of the attributes
#     to be set, especially if an error occures.
#   get
#     Get the value of an attribute from the object. Should be better
#     automatically done by view updates. You can get more than one value
#     in one get.
#     Attributes are specified as option and are returned as option with
#     their values as parameter. Some metaattributes are only viewable, but
#     have no own value, so they return only the attribute as option without
#     parameter/braces. Array attributes are specified as option
#     with the index as parameter and are returned as option with the index
#     as first parameter and its values as second and more parameter.
#     If you don't specify an option indicating the value of an attribute
#     to be returned, all available values of attributes are returned.
#     This can generate large replies with unknown attributes and shouldn't
#     be used in release versions. In case of an array attribute, entries
#     will most likely be omitted without the attribute option. To get all
#     values of an array attribute, specify the array attribute as option
#     without parameters. This can generate really large replies, but this
#     is sometimes the easiest way to get all valid values if the range of
#     indices is unknown. If the reply gets to large, indices can be
#     omitted by the command station. Omitted indices will be indicated by
#     a <attribute> <[> ... <]> listentry. This will not result in an error.
#   create
#     Create an object. A newly created object is a private object until
#     made public with append. Only one private object can exist at a give
#     time. Objects have to be created at their manager object.
#   delete
#     Delete an object. You sometimes need a control to do this.
#   request
#     Request a view or control.
#   release
#     Release a view or control.
#   link
#     Add an object to a manager object.
#   unlink
#     Remove an object from a manager object.
```

queryObjects
Get a list of objects managed by this object.

9.6 Returncodes, errors and warnings

The returncode 0 means ok and the command station has executed the
command. All other return codes mean error and the command station
hasn't executed the command at all.
If the returncode is 0, the errormessage can contain a warning.
If no warning is passed in the error message, the error message is OK.
#

9.6.1 List of implemented returncodes/messages

0 OK
0 OK, but no newline after packet
0 OK, but no attribute
0 OK, control not implemented and not needed
0 OK, but already have control
0 OK, but already free
0 OK, but not controller
0 OK, but no view changed
0 OK, but obsolete option at 0
0 OK, but obsolete attribute at 0
0 OK, but function not exists at 0
0 OK, but function not used at 0
0 OK, but string shortened at 0
0 OK, but wrong fallback at 0
0 OK, but no RailCom port at 0
0 OK, but index not used at 0
10 unknown command at 0
11 unknown option at 0
11 unknown objectclass at 0
11 invalid option at 0
11 size with other arguments
11 append contradicts discard
11 protocol/addr with other attributes
12 invalid parameter at 0
12 no data for switch at 0
12 contradicting parameters at 0
12 unknown parameter at 0
13 invalid character in string at 0
13 NULL in string at 0
15 unknown object at 0
18 internal error
18 internal error at 0
19 invalid character at 0
20 line to long, 0 bytes

```
#   20 to many arguments at 0
#   21 syntax error, '"' expected at 0
#   21 invalid ucs code at 0
#   21 syntax error at 0
#   21 empty parameterlist at 0
#   21 invalid utf-8 coding at 0
#   21 leading zero in decimal at 0
#   21 invalid character in integer at 0
#   21 integer out of range at 0
#   21 syntax error, '(' expected at 0
#   21 syntax error, ')' expected at 0
#   21 syntax error, ']' expected at 0
#   22 function not used at 0
#   22 invalid state at 0
#   22 read only attribute at 0
#   22 const attribute at 0
#   22 attribute can't be used at create at 0
#   22 no queryable attribute at 0
#   22 cannot insert at 0
#   22 cannot remove at 0
#   22 not in list at 0
#   22 not allowed on unappended object at 0
#   22 discard with other options at 0
#   22 max objects reached at 0
#   22 not possible at 0
#   22 not allowed at 0
#   22 bad wifi mode at 0
#   22 invalid object at 0
#   22 not allowed
#   22 no link possible
#   22 no unlink possible
#   22 no managerobject
#   22 invalid address at 0
#   22 invalid protocol at 0
#   22 function not exists at 0
#   23 duplicate option at 0
#   25 control needed
#   25 controlled by somebody else
#   25 couldn't get control
#   28 no object to discard at 0
#   29 no add
#   29 force without control
#   29 update without view
#   29 attribute without view
#   29 listattribute without view
#   29 viewswitch without view
#   29 needs append at 0
#   29 no id
#   29 no state
```

29 no view or control
29 missing addr
29 missing mode
29 missing cv
30 missing funcset-parameter at 0
30 missing addrext-parameter at 0
30 missing switch-parameter at 0
30 missing cv-parameter at 0
30 missing parameter at 0
30 missing bool-parameter at 0
30 missing 0/1-parameter at 0
30 missing integer-parameter at 0
30 missing integer-parameter at 0
30 missing string-parameter at 0
30 missing string-parameter at 0
30 missing string-parameter at 0
30 missing keyword-parameter at 0
30 missing attribute-parameter at 0
30 missing listattribute-parameter at 0
31 not implemented at 0
31 only appending modules implemented at 0
31 no control possible
31 only for last module implemented
32 no RailComPlus without RailCom at 0
32 no go after stop
32 address conflict at 0
35 already created at 0