Network specification for the PC interface

F or P ro t o ll V e rsi on 0 . 2
Third edition, June 2011

1. Lizenzbestimmungen 2
1.1. General 2
1.2. Completion of the license 2
1.3. Prohibited use 2
1.4. Restrictions 2
1.5. Disclaimer of Warranties 2
1.6. Changes to protocol and license terms 2
1.7. Choice of law 2
1.8. Final provision 2
2. Verbindung 3
3. Objekte 3
4. Clients/Teilnehmer 3
5. Nachrichten 3
5.1. Structure of the messages 3
6. Befehle 3
6.1. queryObjects(id, options, ...) 3
6.2. set(id, options, ...) 4
6.3. get(id, options, ...) 4
6.4. create(id, options, ...) 4
6.6. request(id, options, ...) 5
6.7. release(id, options, ...) 5
7. Anhang 6
7.1. Basic object ECoS (id=1) 6
7.2. Basic object programming track (id=5) (in planning) 6
7.3. Basic object LokManager (id=10) 6
7.5. Basic object shuttle train control (id=12) (in planning) 7
7.6. Basic object device manager (id=20) (in planning) 7
7.7. List object Sniffer (id=25) 7
7.8. Feedback manager (id=26) 7
7.9. List object Booster (id=27) (in planning) 7
7.10. Basic object control desk (id=31) (in planning) 7
7.11. Locomotive list object (id=dynamic) 8th
7.13. List object feedback module (id=dynamic) 9
7.14. List object shuttle train route (id=dynamic) 9
8. Werteliste 10
8.1. Locomotive list 10
8.2. Function description list 11

1. License Terms

1.1. General
This document describes procedures for communication between a network participant (hereinafter referred to as "client") and a network service provider (hereinafter referred to as "server"). In the following, these procedures are referred to as "protocol".

Electronic solutions ulm GmbH & Co. KG (hereinafter referred to as "ESU") holds all rights, in particular the copyright of the protocol.

1.2. completion of the license
This license text represents an offer to anyone to conclude a license agreement. The license agreement comes about through the implementation of the protocol on a data processing system. The user is granted non-exclusive, restricted and non-transferable rights of use free of charge.

The implementation of the protocol in client software for PC systems (e.g. home computers or laptops) is permitted without restrictions.

1.3. Prohibited Use
It is expressly forbidden to implement the protocol in a server. In particular, this includes the use of the protocol in a device suitable for controlling model toys.

The licensee is also prohibited from developing further communication processes based on the protocol. This expressly includes the modification of the existing protocol.

In addition, the implementation of the protocol in software that can run on systems other than PCs is prohibited.

1.4. restrictions
The licensee and third parties are prohibited from passing on this document or making it publicly accessible in any other form. Content is licensed, not sold.

1.5. Disclaimer of Warranties
No responsibility or liability is assumed for the accuracy, content, completeness, reliability, operability or suitability for any particular purpose of the log. In particular, no claims can be asserted against ESU for damage caused by the use of the protocol.

The Protocol is provided "as is" without warranty of any kind. No support is guaranteed for the protocol.

1.6. Changes to protocol and license terms
ESU is entitled to change the protocol at any time and without notice. In addition, ESU is entitled to unilaterally change the provisions of this license agreement.

1.7. choice of law
The law of the Federal Republic of Germany applies to all legal relationships between the parties, including tort law. The place of jurisdiction is Ulm.

1.8. final provision
Should individual provisions of this contract be ineffective or unenforceable or become ineffective or unenforceable after the conclusion of the contract, this shall not affect the validity of the rest of the contract. The invalid or unenforceable provision shall be replaced by a valid and enforceable provision whose effects come as close as possible to the economic objective pursued by the contracting parties with the invalid or unenforceable provision.

The above provisions apply accordingly in the event that the contract proves to be incomplete. § 139 BGB does not apply.

2nd connection

TCP socket connection on port 15471.

Receive buffer is 1024 bytes large, commands must not be split. However, there may be several complete commands in one package. If a command cannot be interpreted within the receive buffer, the rest of the receive buffer is discarded.

3. Objects

ECoS is accessed via objects. These objects are addressed by a unique ID. There are numerous base objects with constant IDs that are used to configure and manage other objects. The list objects are managed in the so-called object managers. The LokManager (basic object) with ID 10 manages, for example, all locomotives (list object). The IDs of the list objects can be queried and new list objects can be added via this LokManager.

4. Clients/Participants

The ECoS is a central server. All objects must be registered on this server, ie on the ECoS. When a hand controller is connected to the ECoS, for example, it is automatically registered with the ECoS and is assigned, for example, the locomotives last called up on this hand controller. Each participant can register in two different ways at one or more objects. A control is needed to manipulate an object. A view allows an object to be monitored. A hand controller with a locomotive list of 10 locomotives, for example, will register as a view on all locomotives and as a control on a locomotive that is to be controlled. In the system, only one participant may have control of an object, ie only one participant may manipulate an object at the same time.

By registering a participant in an object as a view, this participant is automatically informed about changes to this object via so-called events. A participant who has registered as a control and as a view on an object is not informed about changes to the object.

5. News

An ASCII protocol is used for communication via the PC interface. Strings such as names are encoded in UTF8 and are always enclosed in double quotes. Quotation marks in strings must be sent twice for better parsing of quotation marks in strings.

Each communication is confirmed by the ECoS with an error code (reply). Messages triggered by the ECoS (events) do not have to be confirmed by the PC. A new line (\n, ASCII=10) is used as the end of the line. Commands can also be preceded by a carriage return (\r, ASCII=13) in front of this new line.

Each message is always addressed to an object identified by a unique ID. There is a series of permanently defined IDs (base objects) and dynamically assigned IDs (list objects). All messages to ID 10 concern the so-called LokManager or ID 1 the basic functions of the ECoS. A list of the fixed IDs of the basic objects can be found in the appendix. 16 bits are used for the IDs and IDs are always unique.

5.1. structure of the messages

Each message from the ECoS to the PC has the form

<HEADER cmd>

TEXT

<END x (str)>

HEADER is either REPLY or EVENT.
A REPLY marks a response to a request via the PC interface, eg requesting the name of a locomotive.

An EVENT is a message generated by the ECoS, such as the information to the PC that the speed of a locomotive has changed.

In addition, the command to which the message refers is repeated with cmd in the header. For an EVENT, cmd contains the ID of the object.

ENDE is always END followed by a numeric error code x and the associated error description str in brackets. Error code 0 (str=OK) means the command ran without errors.

Several lines of additional information can be contained between the header and the end (TEXT). Each line in TEXT has the structure:

id option

id is the ID of the object and option contains additional information. Every

command from the PC to the ECoS has the form

command(id [, options [, ...]])

The options, in turn, can have optional parameters enclosed in square brackets ([]). Up to 10 options are allowed per command. The options options apply to all commands to any ID, although not all options can be interpreted by every object.

6. Commands

A general description of the commands to the ECoS now follows. The first argument is always an ID via which an ECoS object is addressed. Up to 10 further arguments (options) can follow the ID.

The description of the objects explains the options supported by the respective object. For example, the duration option is only supported by an item item object. The following table structure is used in the examples for the individual commands:

Command to the ECoS	ECoS response to this command	comment
description of this example		

6.1. queryObjects(id, options, ...)

This command returns a list of objects that belong to the object with ID id. With the size option, only the number of elements is returned instead of the entire list. With the option nr[min, max] a range of the list can be requested.

Example:

queryObjects(10, name)	<REPLY queryObjects(10, name)> 1000 name["Big Boy"] <END 0 (OK)>	
Object 10 (LokManager) requests a list of the objects. In this case it is the entire locomotive list of the ECoS. In the response to a queryObjects command, the objects are identified by the IDs of the objects. In this example there is only one locomotive with the ID 1000 in the ECoS locomotive list. The returned list of objects can be specified more precisely using options in the queryObjects command. In the example above, the name of the object, in this case the name of the locomotive, is output in addition to the object ID using the name option.		

Example:

queryObjects(10, size)	<REPLY queryObjects(10, size)> 10 size[1] <END 0 (OK)>	
Object 10 (LokManager) requests the number of objects. In this example there is only one locomotive in the ECoS locomotive list.		

Example:

queryObjects(10, no[0,3])	<REPLY queryObjects(10, nr[0, 3])> 1000 1001 1005 1006 <END 0 (OK)>	
Object 10 (LokManager) requests the first 4 entries in the list. In this example there are at least 4 locomotives with the IDs 1000, 1001, 1005 and 1006 in the ECoS locomotive list.		

6.2. set(id, options, ...)
This command can be used to set individual properties of an object. Since at least one property is set, at least two parameters (ID and at least one option) are required for this command. As a rule, each option will also have another parameter in square brackets that contains the new value.

Example:

set(1000, addr[3])	<REPLY set(1000, addr[3])> 1000 addr[3] <END 0 (OK)>	
Address 3 is assigned to the object with ID 1000.		

Example:

set(1000, speed[12])	<REPLY set(1000, speed[12])> 1000 speed[12] <END 0 (OK)>	
The speed option is set to 12 for the object with ID 1000. If the object with ID 1000 is a locomotive, it then runs at speed 12. Speeds are always normalized to 127 speed steps with the ECoS, whereby a speed of 1 means an emergency stop, provided the decoder supports this.		

6.3. get(id, options, ...)
With this command, individual properties of an object are queried.

Example:

get(1000, name, speed)	<REPLY get(1000, name, speed)> 1000 name["Big Boy"] 1000 speeds[12] <END 0 (OK)>	
The name and speed of the object with ID 1000 are requested.		

6.4. create(id, options, ...)
A new object is created. Since the object ID is assigned by ECoS, a new object must be created with the corresponding object manager. A new locomotive is therefore created with an ID 10 (LokManager). The object ID of the object manager must then also be used to configure the newly created object. Thus, only one new object can be configured on an object manager by a participant. A new object can only be created by this participant once the creation of the new object has been completed. The creation of an object is completed with a request(id, append) command. This command assigns a new ID to the object and from this point on the new object is in the list of list objects of the object manager.

Example for creating a locomotive:

create(10)	<REPLY create(10)> 10id[1007] <END 0 (OK)>	It will be a object with the ID 1007 created. this object is for others participant first after one append command visible.
set(1007, addr[5])	<REPLY set(1007, addr[5])> 1007 addr[5] <END 0 (OK)>	
set(1007, name["Big Boy"])	<REPLY set(1007, name["Big Boy"])> 1007 name["Big Boy"] <END 0 (OK)>	
create(10)	<REPLY create(10)> <END 35 (NERROR_NOAPPEND)>	error: the previously generated Lok must first through a append command in the locomotive list recorded become.
create(10, append)	<REPLY create(10, append)> 10 id[1007] <END 0 (OK)>	
create(10, addr[10], name["Test"], protocol [DCC 2 8] , append)	<REPLY create(10, addr[10], name["Test"], protocol[DCC28], append) 10id[1008] <END 0 (OK)>	It will be one locomotive with the address 10, the locomotive name "Test" and the Protocol DCC28 created.
create(10, append)	<REPLY create(10, append)> <END 0 (NERROR_OK>	It will be one new locomotive with default values created.
Since the loco manager is addressed with the create command (ID=10), a new loco is created. Then this locomotive is given the address 5 and the name "Big Boy". Each create command has an append command that adds the new object to the list of objects. Only after this append command can other participants access this newly created object.		

6.5. delete(id, options, ...)
An existing object is deleted. Objects with fixed IDs cannot be deleted. In order to be able to delete an object, the participant must have successfully logged on to this object as a control.

Example:

delete(1005)	<REPLY delete(1000)> <END 25 (NERROR_NOCONTROL)>	Error: the participant has no controls on this Object.
request (1 0 0 5 , control)	<REPLY request(1005, control)> <END 0 (OK)>	
delete(1005)	<REPLY delete(1005)> <END 0 (OK)>	
The object with ID 1005 is deleted. On the first attempt, the participant did not yet have control of this object. When an object is deleted, the control and possibly a view of this object are then automatically deleted.		

6.6. request(id, options, ...)
A client can register with an object in two ways.

If a client registers as a viewer (view option), the client is automatically informed of changes to the respective object by events.

If a client registers as a controller (option control), the client can make changes to the object. For example, if a locomotive is on the ECoS driving screen and is controlled from there (the driving screen participant has control of this locomotive object), a client receives an error message with a request(ID, control). For example, a loco cannot be controlled on the ECoS as long as it is under the control of a manual controller. A client can only get a control for this locomotive when the control for this locomotive is released.

Each client can only register once on an object as a controller and/or viewer. Registration as a viewer is always possible as long as the object exists. Registration as a controller is only possible if no other participant (handheld controller, driving screen on the ECoS, client via the PC interface) has registered as a controller for this object. Unsuccessful registration as a controller is not saved, ie a participant does not automatically receive a control once he has unsuccessfully registered as a control and the object becomes available. The Competitor must try again to register as a Control again. If a participant releases a control on an object, a message is sent to all registered viewers as an event.

With the additional option force, a locomotive can be "stolen" by another participant.

Example:

request(1000, views)	<REPLY request(1000, view)> <END 0 (OK)>	
request(1000, control)	<REPLY request(1000, control)> <END 25 (NERROR_NOCONTROL)>	error: a another participant has a controller on this Object.
	<EVENT 1000> 1000 speeds[40] <END 0 (OK)>	Of the participants, the the Control up this object has, has the speed on the new one value 40 set. This events automatically to all registered Participant shipped that a view on this object have.
request(1000, control, force)	<REPLY request(1000, control, force)> <END 0 (OK)>	the participant, the the Control up this object had, will this object "stolen".
A participant registers with the object with the ID 1000 first as a view. This automatically informs him about changes to this object. An attempt is then made to also get a control on this object. This attempt is acknowledged with an error because another participant (e.g. driving screen on the ECoS) has control over this object. With the additional option force, however, the control is "stolen" from this participant.		

6.7. release(id, options, ...)
Opposite of request. The client logs off as viewer (option view) or controller (option control). All viewers registered to this object receive a message when a participant logs off as a control.

Example:

release(1000, view, controls)	<REPLY release(1000,view,control)> <END 0 (OK)>	
The participant completely logs out of this object. From now on, he will no longer be informed automatically about changes to this object and the participant may no longer make any changes to the object.		

7. Appendix

List of the commands of the PC interface of the ECoS. The corresponding commands and the supported options are listed for each ID.

There are some constant IDs of important base objects. These objects cannot be created or deleted either. It is also not necessary to register as a control on a base object. In a base object, lists are managed via list objects.

7.1. Basic object ECoS (id=1)

request(1, view)	Register as a view with the base object ECoS.
release(1, view)	Log out as a view
delete(1005)	<REPLY delete(1005)> <END 0 (OK)>
set(1, stop)	Corresponds to the stop button on the ECoS
set(1, go)	Corresponds to the Go button on the ECoS
get(1, info)	Information about the ECoS: <REPLY get(1, info)> 1 ECoS 1 ProtocolVersion[0.1] 1 ApplicationVersion[1.0.1] 1 HardwareVersion[1.3] <END 0 (OK)>
get(1, state)	Get current status information: <REPLY get(1, status)> 1 Status[val] <END 0 (OK)> val=STOP val=GO val=SHUTDOWN

7.2. Basic object programming track (id=5) (in planning)

With this basic object, a participant gets access to the programming track of the ECoS via the PC interface. The functionality will be comparable to the decoder programming in the setup menu of the ECoS.

7.3. Basic object LokManager (id=10)

request(10, view)	Register the participant as a view. The participant is automatically informed if something changes in the LokManager. If, for example, a new locomotive is added by another participant, all participants who have registered as a view will be informed of this.
release(10, view)	Log off a participant as a view.
queryObjects(10)	Output of the complete locomotive list. Only the IDs of the existing locomotives are output.
queryObjects(10, name)	Output of the complete locomotive list with IDs and locomotive names.
queryObjects(10, addr)	Output of the complete locomotive list with IDs and address.
queryObjects(10, protocol)	Output of the complete locomotive list with IDs and protocol.
queryObjects(10, addr, Surname)	Output of the complete locomotive list with address and locomotive name.
get(10, size)	It becomes only the Number existing locomotives issued.
queryObjects(10, no[min, max])	The (max-min+1) locomotives are output from the minimum locomotive.
create(10)	A new locomotive is created. However, this is not yet included in the locomotive list and cannot be used by any other participant.
create(10, append)	The newly created locomotive is included in the locomotive list.
create(10, discard)	The newly created locomotive is discarded.
create(10, addr[val])	An address for the new locomotive can also be set with the create command.
create(10, name[str])	A loco name for the new loco can also be specified with the create command.
create(10, protocol[val])	The protocol for the new locomotive can also be set with the create command.
get(10, id)	The ID of the new locomotive to be created is output.
create(10, consist)	New multiple traction.

7.4. Basic object switching item manager (id=11)

request(11, view)	Register the participant as a view. The participant is automatically informed if something changes in the switching item manager. If, for example, a new switching item is created by another participant, all participants who have registered as a view will be informed of this.
release(11, view)	Log off a participant as a view.
queryObjects(11)	Spend the complete list existing switching article.
queryObjects(11, name1)	Spend the complete list existing Switch items with IDs and Name1 (name2 and name3 are also possible).
queryObjects(11, addr)	Spend the complete list existing Switching items with IDs and address (addrext is also possible).
get(11, size)	Only the number of available switching items is output.
queryObjects(11, nr[min, max])	The (max-min+1) switching items are output from the minimum switching item.
create(11)	A new item is created.
create(11, append)	The newly created switch item is added to the list of existing switch items.
create(11, discard)	The newly created switching item is discarded.
create(11, route)	New driveway.

set(11, switch[ProtocolAddrPort])	Setting a port directly. Protocol is either MOT or DCC. Port is either r or g (e.g. set(11, switch[MOT10r])). If there is an object for this address, it is switched according to the programmed addresses. If there is no object for this address, this command is issued to the track. If Protocol is omitted, the default default protocol is used.
get(11, switch[ProtocolAddrPort])	reading the current Attitude one accessory ports. If no object exists for this address and no set command has been sent to this address, an error message is returned. If Protocol is omitted, the default default protocol is used.

7.5. Basic object shuttle train control (id=12) (in planning)

queryObjects(12)	Spend the complete list existing shuttle train routes.
queryObjects(12, name)	Spend the complete list existing Shuttle routes with IDs and names.
queryObjects(12, size)	It becomes only the Number existing Shuttle train routes given.
queryObjects (12, nr[min, max])	The (max-min+1) shuttle train routes are output from the minimum shuttle train route.
create(12)	A new shuttle train route will be created.
create(12, append)	The newly created shuttle train route is added to the list of existing shuttle train routes.
create(12, discard)	The new applied shuttle train route becomes discarded.

7.6. Basic object device manager (id=20) (in planning)

request(20, view)	Register as a view.
release(20, view)	Log out as a view.
queryObjects(20)	It becomes one list all at the ECoSlink connected Devices issued. It will a brief description of the device is also automatically output.
queryObjects(20, size)	Output number of existing devices.
queryObjects(20, nr[min, max])	Output partial list.

This basic object manages the devices in the ECoS system. This will make it possible for a participant to configure devices connected via the PC interface. For example, PC software can influence the locomotive list of the connected hand controllers. When the list is output, a description of the device is sent in addition to the ID, since the list objects managed here can be different list objects.

7.7. List object Sniffer (id=25)

request(25, view)	Register as a view. The participant automatically receives a message about packets that have arrived at the sniffer.
release(25, view)	Log out as a view.

Access to the sniffer should be made directly via this base object. This means that packets that are not linked to a locomotive can also be evaluated by a participant on the sniffer via the PC interface.

This object is also managed as a list object in the DeviceManager base object.

7.8. Feedback manager (id=26)

request(26, view)	Register participants as a view.
release(26, view)	Log out as a view.
queryObjects(26)	Output a list of existing feedback modules.
get(26, size)	Output number of existing feedback modules.
queryObjects(26, nr[min, max])	partial list existing feedback modules spend.
queryObjects(26, ports)	Output of the feedback modules with IDs and number of available ports.
delete(26, del[pos])	S88 module at position pos is deleted. Not available for ECoSDetector.
create(26, add[pos, ports])	Create a new S88 module at position pos. The optional ports parameter sets the number of ports for the S88 module. Not available for ECoSDetector.
set(26, size[val])	New S88 module has val ports.

The feedback modules (s88 and ECoSDetector) are configured via this object. This object is also managed as a list object in the DeviceManager base object, but it is again a base object and manages the existing feedback modules in a list.

The IDs of the S88 modules depend on the position within the S88 bus. The first module always has ID 100, the next one ID 101, etc.

The IDs of the ECoSDetector modules depend on the ECoSDetector number. The ECoSDetector with number 1 has ID 200, with number 2 ID 201 etc.

7.9. List object Booster (id=27) (in planning)

set(27, delay[val])	Sets the short-circuit detection delay for the booster module integrated in the ECoS.
get(27, delay)	Reading out the short-circuit detection delay.

This list object manages the booster module integrated in the ECoS. This module has the delay in short-circuit detection as the only parameter.

This object is also managed as a list object in the DeviceManager base object.

7.10. Basic object control desk (id=31) (in planning)

Direct access to the control panel of the ECoS should be possible via this basic object. In this way, a participant could, for example, configure the assignment of the individual tabs for the switching items on the PC interface.

7.11. List object locomotive (id=dynamic)

request(id, view)	Register participants as a view.
request(id, control)	Register participants as controls.
request(id, control, force)	If another participant already has control of this object, it can be "stolen" by this participant.
release(id,view)	Log out as a view.
release(id, control)	Log out as a control.
get(id,addr)	Output of the locomotive address.
set(id, addr[val])	Setting the locomotive address.
get(id, name)	Output of the locomotive name.
set(id, name[str])	Setting the locomotive name.
get(id, protocol)	Output protocol used.
set(id, protocol[val])	Setting the protocol. val can have the following values: MM14, MM27, MM28, DCC14, DCC28, DCC128, SX32, MMFKT
get(id, profile)	Output decoder profile.
set(id, profile[val])	Set decoder profile. Switching the profile sets CVs to the default values stored in the ECoS.
get(id, sniffer)	Output sniffer address.
set(id, sniffer[addr])	Set sniffer address.
get(id, favorite)	Show whether the locomotive is one of the favorites.
set(id, favorite[val])	Mark the locomotive as a favorite.
get(id, speedindicator)	The speed is displayed in km/h. The speedindicator value indicates the speed in km/h at maximum speed level. If this value is 0, the speed is displayed in speed steps.
set(id, speedindicator[val])	Setting the speed in km/h at maximum speed level.
get(id, cv[nr])	Output CV no.
set(id, cv[nr, val])	Set CV nr to val. The new values are only stored in the ECoS and are not programmed onto the decoder.
get(id,speed)	Output of the current speed of the locomotive (normalized to 127 speed steps).
set(id, speed[val])	Set the speed of the locomotive (normalized to 127 speed steps).
get(id, speedstep)	Spend the current speed in Speed levels depending on the protocol.
set(id, speedstep[val])	Set the current speed in Speed levels depending on the protocol.
get(id, you)	Spend the current Direction the locomotive (1=backwards).
set(id, dir[val])	Set the current Direction the locomotive (1=backwards).
get(id, func[no])	Output the status of function no.
set(id, func[no, val])	Setting the function nr to the value val (for digital outputs, val can be 0 or 1).
set(id, stop)	emergency stop.
link(id, id[id2])	Add a locomotive with ID id2 to the multi-traction or activate the shuttle train control for this locomotive with ID id2.
unlink(id, id[id2])	Delete the locomotive with the id2 from the multi-traction or deactivate the shuttle train control for this locomotive with the ID id2.
delete(id)	Delete this locomotive.
queryObjects(id)	Spend the with this locomotive linked objects (locos in one multitraction or shuttle train control).
get(id, funcdesc[nr])	Query of the function description. Return value: id funcdesc[nr, desc [,moment]], with desc from the function description list. If the function is a moment function, the optional moment parameter is also output.
get(id, funcexists[nr])	Extension of funcdesc. Return value: id funcexists[nr, desc [,moment]], desc if function not available -2, if function not assigned -1
set(id, funcdesc[nr, desc[,moment]])	Set the functional description desc out of Function description list for function no. With the optional parameter moment, the function is assigned as a moment function. With desc=-1 the function is hidden.
get(id, locodesc)	Returns locomotive symbol. • id symbol[locotype, imagetype, imageindex] with locotype from the values LOCO_TYPE_E, LOCO_TYPE_DIESEL, LOCO_TYPE_STEAM and LOCO_TYPE_MISC as a fallback if imageindex (see below) cannot be used. • imagetype from the values IMAGE_TYPE_INT (system images) and IMAGE_TYPE_USER (images transferred from the user to the device) • imageindex from the locomotive image list, imageindex and locotype are permanently linked in the device.
set(id, locodesc[imagetype, imageindex])	Sets the locomotive symbol.

7.12. List object switching article (id=dynamic) (in planning)

request(id, view)	Register participants as a view.
request(id, control)	Register participants as controls.
request(id, control, force)	If another participant already has control of this object, it can be "stolen" by this participant.

release(id,view)	Log out as a view.
release(id, control)	Log out as control.
get(id, state)	Returns the current status of the switch item.
set(id, state[val])	Sets the status of the switch item to val.
get(id,addr)	Output of the address of the switching article.
get(id, addrext)	Output extended addressing. Example: addrext[3g, 3r, 4g, 4r] This accessory is connected with a drive at address 3 and with a drive at address 4. If a drive is to be reversed, the address must be set with addrext[3r, 3g, 4g, 4r]. If extended addressing was programmed, the extended addressing is output with a get(id, addr). If no extended addressing is used, the 4 ports are set to the corresponding default values (addr[3] then results in addrext[3g, 3r, 4g, 4r].
set(id, addr[val])	setting the address.
set(id, addrext [val1, val2, val3, val4])	Advanced addressing. Up to 4 ports of a decoder can be programmed per solenoid.
get(id, protocol)	Output the protocol used.
set(id, protocol[val])	Set protocol (val either MM or DCC)
get(id, name1)	Print the first line of the name.
set(id, name1[str])	Setting the first line of the name.
get(id, name2)	Print the second line of the name.
set(id, name2[str])	Setting the second line of the name.
get(id, name3)	Print the third line of the name.
set(id, name3[str])	Setting the third line of the name.
get(id, symbol)	Output the symbol.
set(id, symbol[no])	Setting the icon.
get(id, mode)	Output whether switching item is impulse (PULSE) or switching (SWITCH).
set(id, mode[val])	Set of switching behavior (PULSE or SWITCH).
get(id, duration)	Output of the switching duration.
set(id, duration[val])	Set the switching duration (val either 250, 500, 750, 1000 or 2500).
link(id, id[id2], state[val])	Add switching item with ID id2 and status val to this route. If state is not specified, the current state of the object with ID id2 is used.
unlink(id, id[id2])	Remove switching items with ID id2 from this route.
delete(id)	Delete switching article.
queryObjects(id)	Spend the to this driveway belonging switching article.

7.13. List object feedback module (id=dynamic)

request(id, view)	Register participants as a view. Only 100 (s88) / 200 (ECoSDetector).
release(id,view)	Log out as a view.
get(id, ports)	Output port number.
set(id, ports[val])	Set port count to 8 or 16. s88 modules only (IDs 100-163).
delete(id)	Delete S88 module. s88 modules only (IDs 100-163).
get(id, state)	Returns the current status of the feedback module.
get(id, railcom[port])	delivers the from the ECoSDetector determined Address and direction of the loco on the feedback section (port, address, direction). In the case of an unoccupied section or non-RailCom-capable ECoSDetector inputs becomes 0 reported back.

7.14. List object shuttle train route (id=dynamic)

get(id, station[no])	Output feedback information of station nr (nr can be 1 or 2).
set(id, station[no, m:p])	Station nr has S88 module m and S88 port p (nr can be 1 or 2).
get(id, name)	Output the name of the shuttle train route.
set(id, name[str])	Set the name of the shuttle train route.
get(id,delay)	spend length of stay.
set(id, delay[val])	set length of stay.
delete(id)	Delete shuttle route.

8. Value Lists

8.1. locomotive image list

Type

s Steam

d Diesel

e Electric

m Misc

index	Type	Description
000	s	symbol
001	i.e	symbol
002	e	symbol
003	m	symbol
004	i.e	abj003
005	i.e	abj004
006	i.e	bb66000
007	i.e	br212
008	i.e	br218
009	i.e	br219
010	i.e	br232
011	i.e	br648
012	i.e	cc40100
013	i.e	cc72000
014	i.e	class008
015	i.e	desiro
016	i.e	me026
017	i.e	picasso
018	i.e	ram tea
019	i.e	rail zeppelin
020	i.e	svt137
021	i.e	v060
022	i.e	v080
023	i.e	v188
024	i.e	v200
025	i.e	v212
026	i.e	v216
027	i.e	v300
028	i.e	v320
029	i.e	vt008
030	i.e	vt0115
031	i.e	vt098
032	i.e	vt208
033	e	br103
034	e	br103_red
035	e	br110
036	e	br111
037	e	br143
038	e	br151
039	e	br151_red
040	i.e	br160
041	e	br185
042	e	e044
043	e	e069
044	e	e080
045	e	e091
046	e	e094
047	e	et065
048	e	et087
049	e	ice1
050	e	ice3
051	e	crocodile
052	e	oebb1012
053	e	thalys
054	m	car
055	m	car
056	m	consist_diesel
057	m	consist_e
058	m	consist_s
059	m	controlcar
060	m	crane truck
061	m	silverling
062	s	bigboy
063	s	br001
064	s	br003_streamline_black

065	s	br003_streamline_red
066	s	br024
06	s	br038
068	s	br044
069	s	br050
070	s	br051
071	s	br055
072	s	br064
073	s	br075
074	s	br078
075	s	br080
076	s	br085
077	s	br092
078	s	br096
079	s	br098
080	s	glass box
081	i.e	head

8.2. Feature Description List

Value	Description
0002	function
0003	light
0004	light_0
0005	light_1
0007	sound
0008	music
0009	announce
0010	routing_speed
0011	abv
0032	coupler
0033	steam
0034	panto
0035	high beam
0036	bell
0037	horn
0038	whistle
0039	door_sound
0040	fan
0042	shovel_work_sound
0044	shift
0260	interior_lighting
0261	plate_light
0263	brake sound
0299	crane_raise_lower
0555	hook_up_down
0773	wheel_light
0811	turn
1031	steam blow
1033	radio_sound
1287	coupler_sound
1543	track_sound
1607	notch_up
1608	notch_down
2055	thunderer_whistle
3847	buffer_sound