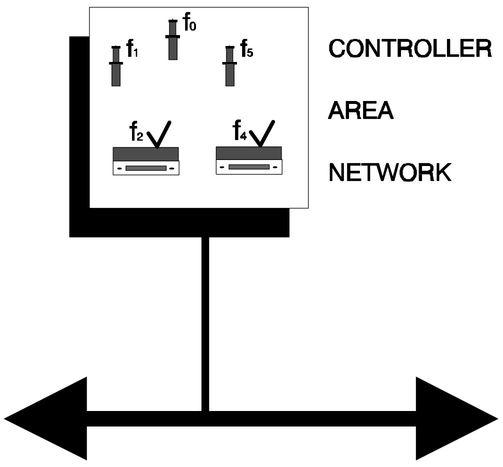
CAN in Automation (CiA)
International Users and Manufacturers Group e.V.



CAN Application Layer for Industrial Applications
CiA/DS202-1
February 1996

CMS Service Specification

1. SCOPE

This document contains the service specification of the CAN based Message Specification (CMS). CMS is part of the CAN Application Layer. This document is part of a set of documents that standardize the CAN Application Layer for Industrial Applications.

2. REFERENCES

/1/: CiA/DS201, CAN Reference Model

/2/: CiA/DS202-2, CMS Protocol Specification

/3/: CiA/DS202-3, CMS Data Types and Encoding Rules

/4/: CiA/DS207, Application Layer Naming Conventions

3. GENERAL DESCRIPTION

3.1 CMS Perspective

CMS is one of the application layer service entities of the CAN Reference Model, see /1/.

3.2 CMS Objects and Services

CMS is a language to specify what COB's a module uses and how they are formatted. CMS can describe all CAN layer 2 features. This means also that existing applications can be described in CMS. Furthermore CMS offers the application a possibility to model its behaviour in the form of objects and remote services on these objects. This allows other applications to cooperate with it by executing these services that CMS supports on these objects. The different service types and primitives are defined in /1/. The notation that is used to describe the CMS Services is also explained in /1/.

3.3 CMS Clients and Servers

An example is given in fig. 1 where CMS is used to model the control of a light switch. The *server* of the switch "physically" interacts with the switch to put on the light. The server "translates" this behaviour into the CMS language e.g a CMS variable with access type 'Write_Only' and data type BOOLEAN. The *client* "logically" interacts with the switch by using the remote CMS 'Write Variable' service.

CMS allows for an object to have one or more servers and zero or more clients, depending on what the object models.

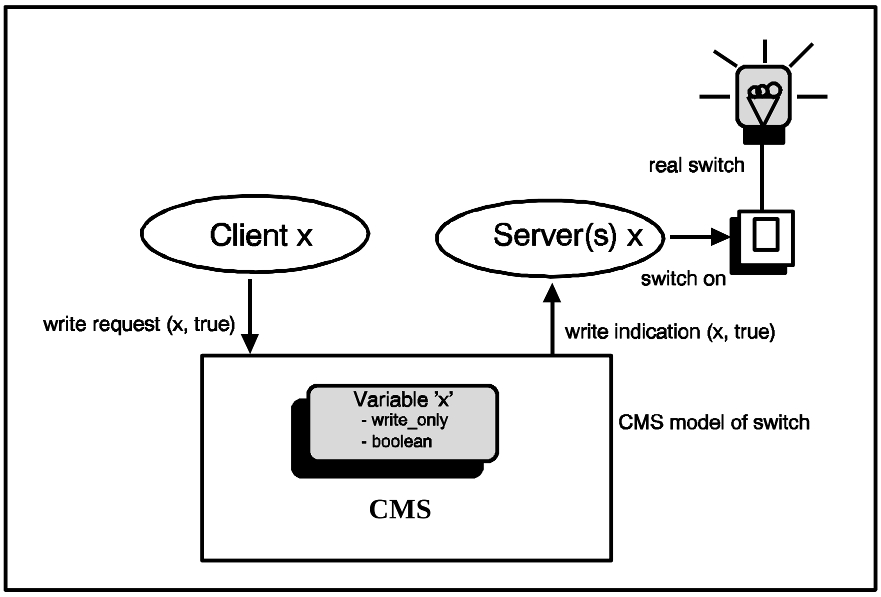


Fig. 1: A CMS Model for a light switch

3.4 CMS Data Types

In order to give a server sufficient information on what he has to do, the client may have to exchange data with the server, e.g the required and current position of the valve. CMS models this by the concept of data types. CMS defines a number of basic types such as INTEGER(5). It is also possible to construct a compound type by collecting basic types in an ARRAY or a STRUCTURE. CMS defines also a number of extended data types.

CMS defines a transfer syntax that describes how values of a particular data type have to be transmitted over the CAN network. The data types and transfer syntax is described in the CMS Data Types and Encoding Rules, see /3/.

3.5 CMS Object Priorities

The priority of a CMS object indicates its importance relative to other CMS objects and is used as an arbitration value by the Medium Access Control of CAN. CMS defines eight priorities in the range [0, 7]. 0 is the highest, 7 the lowest priority.

Priorities can be assigned by the application or the Distributor Service Element (see /1/). In case the Distributor Service Element assigns a priority, the Network Management Service Element (see /1/) controls when the assignment takes places.

3.6 CMS Object Inhibit Times

To guarantee that no starvation on the network occurs for CMS objects with low priorities, CMS objects can be assigned an inhibit time. The inhibit-time of a CMS object defines the minimum time that has to elapse between two consecutive invocations of a CMS remote service for that CMS object.

Inhibit-times can be assigned by the application or the Distributor Service Element (see /1/). In case the Distributor Service Element assigns an inhibit-time, the Network Management Service Element (see /1/) controls when the assignment takes places.

3.7 CMS Service Descriptions

The CMS services are described in a tabular form that contains the parameters of each service primitive that is defined for that service. The primitives that are defined for a particular service determine the service type (e.g unconfirmed, confirmed, etc.). How to interprete the tabular form and what service types exist is defined in /1/. In the service descriptions, [a, b] denotes the range of integers from a to b with a and b included. If a > b, the range is empty.

All services assume that no failures occur in the Data Link and Physical Layer of the CAN network. These failures are resolved by the Network Management Service Element, see /1/.

CMS executes a protocol to implement the services on a CMS object. All protocols are defined in /2/. Each protocol needs one or two COB's to be able to transmit and receive data over the CAN network. This document specifies for each CMS service the used COB's and their attributes:

- the COB-Class. There are 8 COB-Classes. A COB-Class relates the number of (remote) receivers and (remote) transmitters for that COB. The Distributor

COB Class	*#receivers*	*#transmitters*
0	0 .. 1	0 .. 1
1	1	0 .. 1
2	≥ 1	0 .. 1
3	≥ 0	0 .. 1
4	0 .. 1	1
5	1	1
6	≥ 1	1
7	≥ 0	1

Service Element checks for each COB whether the total number of (remote) transmitters and (remote) receivers matches the COB-Class.

- the COB-type for both the server- and client:

rx	= receives a COB
tx	= transmits a COB
RTR-rx	= receives the data of a remote COB
RTR-tx	= transmits the data of a remote COB

- the COB-length. If '*' is specified it means that the data length is determined by the CMS Encoding Rules (see /3/), the data- and error type attribute of the CMS object, and the CMS protocol that this COB is used for (see /2/). If a number is specified it means that the COB has a fixed length.

4. VARIABLES

4.1 Attributes

The following attributes are defined for variables:

- name:	see /4/
- user_type:	one of the values {Client, Server}
- priority:	a value in the range [0, 7]
- inhibit-time:	n*100 usec, n $\geq$ 0
- data_type:	see /3/
- error_type:	see /3/
- class:	one of the values {Basic, Multiplexed}
- access_type:	one of the values {Read_Only, Write_Only, Read_Write}

A variable whose class is 'Basic' is called a basic variable. A variable whose class is 'Multiplexed' is called a multiplexed variable. A multiplexed variable is multiplexed with other multiplexed variables into a variable set.

The following attributes are only defined for multiplexed variables:

- var_set_name:	see /4/
- mutiplexor:	a value in the range [0, 127]

The multiplexor is a natural number that uniquely identifies the variable within the variable set. Multiplexed variables inherit the values of the user_type, access_type, priority, and inhibit-time attributes from the corresponding attributes of the variable set. Hence, all multiplexed variables within one variable set have the same value for these attributes. The following attributes are defined for variable sets:

- name:	see /4/
- user_type:	one of the values {Client, Server}
- priority:	a value in the range [0, 7]
- inhibit-time:	n*100 usec, n $\geq$ 0
- access_type:	one of the values {Read_Only, Write_Only, Read_Write}

4.2 Usage

The access type of a variable is seen from the point of view of the client. Variables with access_type 'Read_Only' can be used by a client only to collect data. For basic variables the collected data will be the data that was set by the server in the last 'update variable' service it executed. Data from previous updates is lost. For multiplexed variables the server has to supply the data when requested by the client.

Variables with access_type 'Write_Only' can be used by a client to request one or more servers to execute a command. The client will not know the result of the command execution.

Variables with access_type 'Read_Write' can be used by a client to collect the 'current data' from the server (read service) or to request a server to execute a command and be informed about the result of the command execution (write service).

Variable sets can be used to "multiplex" several variables. All these multiplexed variables will then be mapped onto the COB's that are used by that variable set. This reduces the number of COB's. Within a variable set the variables are identified by a unique "multiplexor".

4.3 Local Services

Define Variable

This service creates a variable with the requested attributes. Var_set_name must have been defined as a variable set. The attributes must not cause the data length of the used COB's to overflow the maximum.

Parameter	*Request*
Argument	**Mandatory**
var_name	mandatory
data_type	mandatory
error_type	optional
class	mandatory
basic_var	selection
user_type	mandatory
acess_type	mandatory
priority	optional
inhibit-time	optional
mux_var	selection
var_set_name	mandatory
multiplexor	mandatory

- **NOTE:** The cooperating applications are responsible for using consistent attributes for the client and the server of the variable.

Define Variable Set

This service creates a variable set with the requested attributes.

Parameter	*Request*
Argument	**Mandatory**
var_set_name	mandatory
user_type	mandatory
access_type	mandatory
priority	optional
inhibit-time	optional

- **NOTE:** The cooperating applications are responsible for using consistent attributes for the client and the server of the variable set.

Update Variable

Through this service the server of var_name updates the value of var_name. Previously updated values for var_name are lost. Var_name must be a basic variable with access_type 'Read_Only' and user_type 'Server' and value must match the data_type attribute of var_name.

Parameter	*Request*
Argument	**Mandatory**
var_name	mandatory
value	mandatory

4.4 Remote Services

Write Variable

Through this service the client of var_name supplies a value to the server(s) of var_name. Var_name must be a variable with access_type 'Write_Only' or 'Read_Write'. The supplied value must match the data_type attribute of var_name.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
var_name	mandatory	
value	mandatory	
Remote Result		**Mandatory**
success		selection
failure		selection
reason		optional

- **Write_Only variables:** The service is unconfirmed. The supplied value is indicated to the server. There are no Response/Confirm primitives. There can be at most one client. There must be at least one server.

- **Read_Write variables:** The service is confirmed. The supplied value is indicated to the server. The Remote Result parameter will indicate the success or failure of the request. In case of a failure, optionally a value of the error_type attribute of var_name confirms the reason. There can be at most one client. There must be exactly one server.

Read Variable

Through this service the client of var_name requests the server of var_name to supply its value. Var_name must be a variable with access_type 'Read_Only' or 'Read_Write'. The supplied value must match the data_type attribute of var_name.

- **Read_Only basic variables:** The service is confirmed. The Remote Result parameter will confirm the requested value as set by the last Update Variable service. There can be zero or more clients. There must be exactly one server.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
var_name	mandatory	
Remote Result		**Mandatory**
value		mandatory

- **Read_Write variables, Read_Only multiplexed variables:** The service is confirmed. The Remote Result parameter will indicate the success or failure of the request. In case of success, the requested value is confirmed. In case of a failure, optionally a value of the error_type attribute of var_name confirms the reason. There can be at most one client. There must be exactly one server.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
var_name	mandatory	
Remote Result		**Mandatory**
success		selection
value		mandatory
failure		selection
reason		optional

4.5 Used COB's

- Read_Only Basic Variable

Client COB-Type	Server COB-Type	COB-Class	COB-Length
RTR-rx	RTR-tx	7	*

- Read_Only Multiplexed Variable

Client COB-Type	Server COB-Type	COB-Class	COB-Length
tx	rx	1	1
rx	tx	4	*

- Write_Only Variables

Client COB-Type	Server COB-Type	COB-Class	COB-Length
tx	rx	2	*

- Access_Type = Read_Write:

Client COB-Type	Server COB-Type	COB-Class	COB-Length
tx	rx	1	*
rx	tx	4	*

5. DOMAINS

5.1 Attributes

- name:	see /4/
- user_type:	one of the values {Client, Server}
- class:	one of the values {Basic, Multiplexed}
- priority:	a value in the range [0, 7]
- inhibit-time:	n*100 usec, n >> 0

For a domain there can be at most one client and there must be exactly one server. A domain whose class is 'Basic' is called a basic domain. A domain whose class is 'Multiplexed' is called a multiplexed domain. The following attribute is only defined for multiplexed domains:

- mux_data_type:	see section 4.2

5.2 Usage

Basic domains can be used to transfer an arbitrary large block of data from a client to a server and vv. The contents of a data block is application specific and does not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications.

Multiplexed domains can be used to transfer multiple *data sets* (each containing an arbitrary large block of data) from a client to a server and vv. The client can control via a *multiplexor* which data set is to be transferred. This multiplexor is a value of a CMS Data Type. The CMS Data Type of the multiplexor and the contents of the data sets are application specific and do not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications.

A domain is transferred as a sequence of *segments*. Prior to transferring the segments there is an initialization phase where client and server can prepare themselves for transferring the segments. For multiplexed domains, it is also possible to transfer a data set during the initialization phase. This mechanism is called an *expedited* transfer.

It is always the client that takes the initiative for a transfer. Both the client and the server can take the initiative to abort the transfer of a domain. By using the *segmented* services, the application will be responsible for the segmentation of the domain. By using the *non-segmented* services, CMS will be responsible for the segmentation.

5.3 Local Services

Define Domain

Parameter	*Request*
Argument	**Mandatory**
dom_name	mandatory
user_name	mandatory
priority	optional
inhibit-time	optional
class	mandatory
basic_dom	selection
mux_dom	selection
mux_data_type	mandatory

This service creates a domain with the requested attributes.

- **NOTE:** The cooperating applications are responsible for using consistent attributes for the client and the server of the domain.

5.4 Remote Services (non-segmented)

When using these services, CMS will be responsible for transferring the domain as a sequence of segments.

Domain Download

Through this service the client of dom_name downloads data to the server of dom_name. The data and optionally its size are indicated to the server. For multiplexed domains the multiplexor of the data set that has been downloaded is indicated to the server. The value of multiplexor must match the mux_data_type attribute of dom_name.

The service is confirmed. The Remote Result parameter will indicate the success or failure of the request. In case of a failure, optionally the reason is confirmed.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
dom_name	mandatory	
data	mandatory	
size	optional	
basic_dom	selection	
mux_dom	selection	
multiplexor	mandatory	
Remote Result		**Mandatory**
success		selection
failure		selection
reason		optional

Domain Upload

Through this service the client of dom_name uploads data from the server of dom_name. For multiplexed domains the multiplexor of the data set that has to be uploaded is indicated to the server. The value of multiplexor must match the mux_data_type attribute of dom_name.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
dom_name	mandatory	
basic_dom	selection	
mux_dom	selection	
multiplexor	mandatory	
Remote Result		**Mandatory**
success		selection
data		mandatory
size		optional
failure		selection
reason		optional

The service is confirmed. The Remote Result parameter will indicate the success or failure of the request. In case of a failure, optionally the reason is confirmed. In case of success, the data and optionally its size are confirmed.

5.5 Remote Services (segmented)

When using these services, the application will be responsible for transferring the domain as a sequence of segments.

Initiate Domain Download

Through this service the client of dom_name requests the server of dom_name to prepare for downloading data to the server. Optionally the size of the data to be downloaded is indicated to the server.

For multiplexed domains the multiplexor of the data set whose download is initiated and the transfer_type are indicated to the server. The value of multiplexor must match the mux_data_type attribute of dom_name. In case of an expedited download, the data of the data set identified by multiplexor is indicated to the server.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
dom_name	mandatory	
size	optional	
basic_dom	selection	
mux_dom	selection	
multiplexor	mandatory	
transfer_type	mandatory	
normal	selection	
expedited	selection	
data	mandatory	
Remote Result		**Mandatory**
success		mandatory

The service is confirmed. The Remote Result parameter will indicate the success of the request. In case of a failure, an abort domain transfer request must be executed. In case of a successful expedited download of a multiplexed domain, this service concludes the download of the data set identified by multiplexor.

Download Domain Segment

Through this service the client of dom_name supplies the data of the next segment to the server of dom_name. The segment data and optionally its size are indicated to the server. The continue parameter indicates the server whether there are still more segements to be downloaded or that this was the last segment to be downloaded.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
dom_name	mandatory	
data	mandatory	
size	optional	
continue	mandatory	
more	selection	
last	selection	
Remote Result		**Mandatory**
success		mandatory

The service is confirmed. The Remote Result parameter will indicate the success of the request. In case of a failure, an abort domain transfer request must be executed. In case of success, the server has accepted the segment data and is ready to accept the next segment. There can be atmost one Download Domain Segment service outstanding for a Domain.

For basic domains a successful 'Initiate Domain Download' service must have been executed prior to this service. For multiplexed domains a successful 'Initiate Domain Download' service with transfer_type 'normal' must have been executed prior to this service.

Initiate Domain Upload

Through this service the client of dom_name requests the server of dom_name to prepare for uploading data to the client. For multiplexed domains the multiplexor of the data set whose upload is initiated is indicated to the server. The value of multiplexor must match the mux_data_type attribute of dom_name.

The service is confirmed. The Remote Result parameter will indicate the success of the request. In case of a failure, an abort domain transfer request must be executed. In case of success, optionally the size of the data to be uploaded is confirmed. In case of successful expedited upload of a multiplexed domain, this service concludes the upload of the data set identified by multiplexor and the corresponding data is confirmed.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
dom_name	mandatory	
basic_dom	selection	
mux_dom	selection	
multiplexor	mandatory	
Remote Result		**Mandatory**
success		mandatory
size		optional
basic_dom		selection
mux_dom		selection
multiplexor		mandatory
transfer_type		mandatory
normal		selection
expedited		selection
data		mandatory

Upload Domain Segment

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
dom_name	mandatory	
Remote Result		**Mandatory**
success		mandatory
data		mandatory
size		optional
continue		mandatory
more		selection
last		selection

Through this service the client of dom_name requests the server of dom_name to supply the data of the next segment. The continue parameter indicates the client whether there are still more segements to be uploaded or that this was the last segment to be uploaded. There can be atmost one Upload Domain Segment service outstanding for a Domain.

The service is confirmed. The Remote Result parameter will indicate the success of the request. In case of a failure, an abort domain transfer request must be executed. In case of success, the segment data and optionally its size are confirmed.

For basic domains a successful 'Initiate Domain Upload' service must have been executed prior to this service. For multiplexed domains a successful 'Initiate Domain Upload' service with transfer_type 'normal' must have been executed prior to this service.

Abort Domain Transfer

Parameter	*Request / Indication*
Argument	**Mandatory**
dom_name	mandatory
reason	optional

This service aborts the up- or download of dom_name. Optionally the reason is indicated. The service is unconfirmed. The service may be executed at any time by both the client and the server of dom_name. If the client of dom_name has a confirmed service outstanding, the Abort Indication is taken to be the Confirm of that service.

5.6 Used COB's

Client COB-Type	Server COB-Type	COB-Class	COB-Length
tx	rx	1	8
rx	tx	4	8

6. EVENTS

6.1 Attributes

- name:	see /4/
- user_type:	one of the values {Client, Server}
- class:	one of the values {Controlled, Uncontrolled, Stored}
- data_type:	see /3/
- priority:	a value in the range [0, 7]
- inhibit-time:	n*100 usec, n >> 0

An event whose class is 'Controlled' is called a controlled event. An event whose class is 'Uncontrolled' is called an uncontrolled event. An event whose class is 'Stored' is called a stored event. The following attributes are only defined for controlled events:

- error_type:	see /3/
- control_state:	one of the values {Enabled, Disabled}

6.2 Usage

An event can be used to model asynchronous behaviour such as a temperature exceeding a certain limit. The occurrence of an event is detected by the server and can be *notified* to the client(s). An event has a data_type attribute to supply additional information about what caused the event to occur such as the actual temperature that exceeded the limit.

Uncontrolled events can be used to implement events that are notified to any client that is "interested". Uncontrolled events are always notified when they occur.

Controlled events can be used to implement an event that can be notified to at most one client. The client can control whether the server notifies the event when it occurs.

Stored events can be used by a server to *store* locally a value of the data_type attribute of an event and optionally notify the client(s). A client, on his own initiative, can *read* the last value of an event that was stored by the server. Previously stored values are lost.

6.3 Local Services

Define Event

Parameter	*Request*
Argument	**Mandatory**
event_name	mandatory
data_type	mandatory
class	mandatory
controlled	selection
error_type	optional
uncontrolled	selection
stored	selection
user_type	mandatory
proiority	optional
inhibit-time	optional

This service creates an event with the requested attributes. The control state of a controlled event will be 'Disabled'. The attributes must not cause the data length of the used COB's to overflow the maximum.

- **NOTE:** The cooperating applications are responsible for using consistent attributes for the client and server of the event.

6.4 Remote Services

- **Event Class = Controlled:** There can be at most one client. There must be exactly one server.
- **Event Class = Uncontrolled:** There can be zero or more clients. There can be at most one server.
- **Event Class = Stored:** There can be zero or more clients. There must be exactly one server.

Notify Event

Through this service the server of event_name notifies the client(s) of event_name that the event has occurred and supplies its value. Event_name must be an uncontrolled event or a controlled event with control_state 'Enabled'. Value must match the data_type attribute of event_name. The service is unconfirmed.

Parameter	*Request / Indication*
Argument	**Mandatory**
event_name	mandatory
value	mandatory

Store Event

Through this service the server of event_name stores the value of event_name. Previously stored values for event_name are lost. Optionally the server immediately notifies this value to the client(s) of event_name.

Parameter	*Request / Indication*
Argument	**Mandatory**
event_name	mandatory
value	mandatory
immediate_notify	optional

Event_name must be a stored event. Value must match the data_type attribute of event_name. The service is local unless immediate notification is requested. In that case the service is unconfirmed.

Read Event

Through this service the client of event_name requests the server of event_name to supply the value as stored by the last Store Event service. The service is confirmed. The Remote Result parameter will confirm the value.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
event_name	mandatory	
Remote Result		**Mandatory**
value		mandatory

Event_name must be a stored event. Value must match the data_type attribute of event_name.

Set Event Control State

Through this service the client of event_name requests the server of event_name to set its control_state to the selected value. Event_name must be a controlled event. The service is confirmed. The Remote Result parameter will indicate the success or failure of the request. In case of a failure, optionally a value of the error_type attribute of event_name confirms the reason.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
event_name	mandatory	
control_state	mandatory	
enabled	selection	
disabled	selection	
Remote Result		**Mandatory**
success		selection
failure		selection
reason		optional

6.5 Used COB's

- Uncontrolled Events:

Client COB-Type	Server COB-Type	COB-Class	COB-Length
rx	tx	3	*

- Controlled Events:

Client COB-Type	Server COB-Type	COB-Class	COB-Length
tx	rx	1	1
rx	tx	3	*

- Stored Events:

Client COB-Type	Server COB-Type	COB-Class	COB-Length
RTR-rx	RTR-tx	7	*