System Specifications

3

Application Environment

4

Application Interface Layer

1

Summary

This Chapter specifies the structure and functioning of servers for the Objects which form the interface between the Application Layer and the Application and Management.

Version 01.01.02 is a KNX Approved Standard.

This document is part of the KNX Specifications v2.1.

Document updates

Version	Date	Modifications
AS v1.0	2001.12.19	Preparation of the Approved Standard.
WD v1.x	2005.08.18	• Supplement 22 “KNX Radio Frequency” Integration of concept of Function Properties.
	2006.04.07	Integration of Reduced Interface Objects.
	2007.01.12	• AN038 “Function Services” Update, completion and correction of Function Properties. Common 1 s reaction time of Function Properties.
	2007.01.15	• AN039 “Property Datatypes” added reference.
	2007.08.21	Restructernig of the specification of the Properties.
	2008.02.08	• AN099 “Interface Objects and Properties in Profiles” added general requirements.
	2008.05.07	• AN072 “A_Authorize” added use of access levels for Authorization.
	2008.08.12	• AN101 “File Transfer Protocol” integrated: File Server
	2008.09.04	• AN101 “File Transfer Protocol” integrated: File Formats.
AS v1.1	2009.01.05	Finalisation of the Approved Standard v1.1.
AS v1.01.01	2011.09.16	Minor editorial corrections.
01.01.02	2013.10.28	Editorial updates for the publication of KNX Specifications 2.1.

References

[01] Chapter 3/3/4 “Transport Layer”
[02] Chapter 3/3/7 “Application Layer”
[03] Part 3/5 “Management”
[04] Chapter 3/5/1 “Resources”
[05] Chapter 3/5/2 “Management Procedures”
[06] Chapter 3/6/3 “External Message Interface”
[07] Chapter 3/7/1 “Interworking Model”
[08] Chapter 3/7/2 “Datapoint Types”
[09] Chapter 3/7/3 “Standard Identifier Tables”
[10] Volume 6 “Profiles”
[11] Volume 7 “Application Descriptions”

Filename: 03_04_01 Application Interface Layer AS v01.01.02.docx
Version: 01.01.02
Status: Approved Standard
Savedate: 2013.10.28
Number of pages: 29

Contents

1 Overview

The Application Interface Layer shall be the layer between the Application Layer and the Application. It shall contain the communication relevant tasks of the application. It shall ease the communication task of the application by offering a communication interface that abstracts from many Application Layer details.

The KNX System allows single-processor and dual-processor device designs. A dual processor device uses additional services to communicate via a serial External Message Interface (EMI) with the external User Application running in the second processor. The EMI is specified in [06].

The following clauses define the client and server functionality and the communication interface of the internal user application located in the BAU.

The Application Interface Layer shall contain the following objects and the communication modes to them. For the specification of the communication modes, please refer to [01].

- **Group Objects**

Group Objects shall be accessible via TSAPs and ASAPs via Application Layer services on

point-to-multipoint, connectionless (multicast) communication mode

Group Objects may also be references to Interface Objects.

- **Interface Objects**

Interface Objects shall be accessible via Application Layer services on

point-to-point connectionless communication mode, and

point-to-point connection-oriented communication mode, and

point-to-domain connectionless (broadcast) communication mode.

The Interface Objects are classified as System Interface Object or Application Interface Object.

- ***System Interface Objects***

This class of Interface Objects is relevant for network management and device management (see [05]). The System Interface Objects are specified in [04].

EXAMPLES the Device Object,
the Group Address Table Object,
the Group Object Association Table Object
the Application Object

- ***Application Interface Objects***

These shall be Interface Objects defined in the user application. They may be defined by the internal or external user application, based on Interface Object structure rules specified in this document. Application Interface Objects may also be referenced by a Group Object reference.

The following clauses explain the data structures of each of the Application Interface Layer objects. Additionally they define by which Application Layer services these objects shall or may be accessible. Both the object client and object server functionality may be implemented by the external or the internal Application Interface Layer. It is recommended to locate the Group Objects and the Interface Objects in the internal Application Interface Layer.

2 Object models

In the KNX System, two different kinds of objects are supported for operational exchanges:

- Group Objects [1)]

  These objects shall serve for a communication model named “Shared Variable Model”.

- Interface Objects

  These objects shall serve for a communication model according the client / server model and - if they are referenced by Group Objects - also the Shared Variable Model of the Group Objects

An application can use each kind of objects at any time.

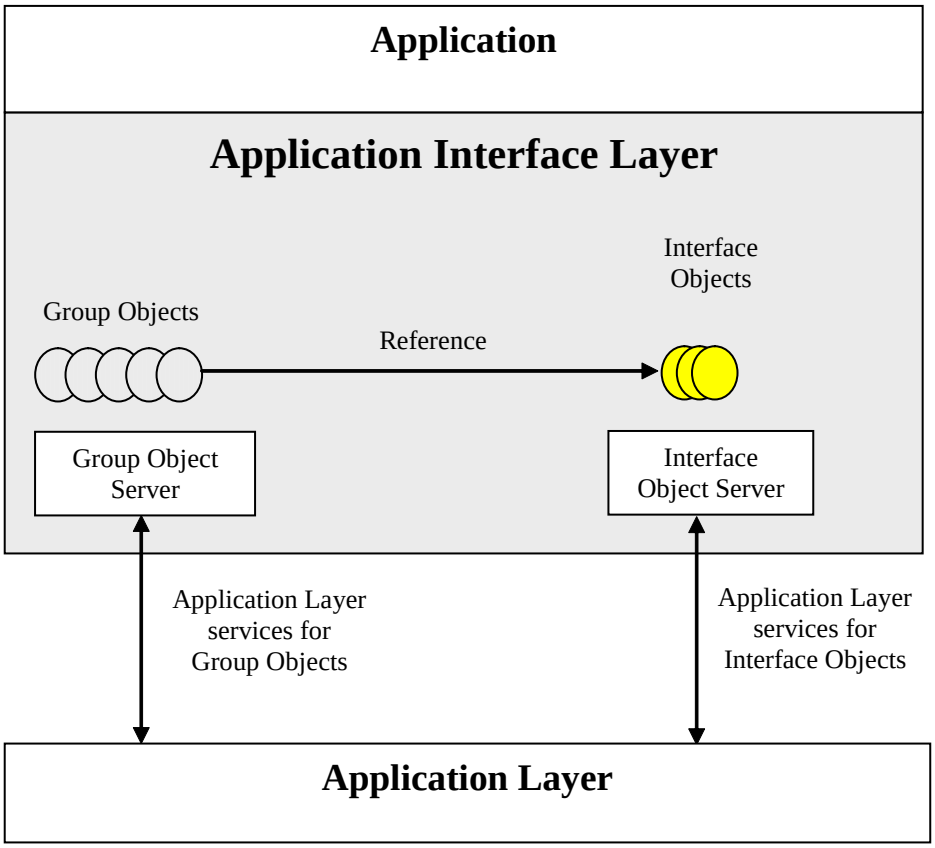


Figure 1 – AIL model

[1)] Group Objects are also accessible via the External Message Interface with a special set of services (A_Value_Read, A_Value_Write, A_Flags_Read and A_Flags_Write) that are specified in [06].

3 Group Object Server

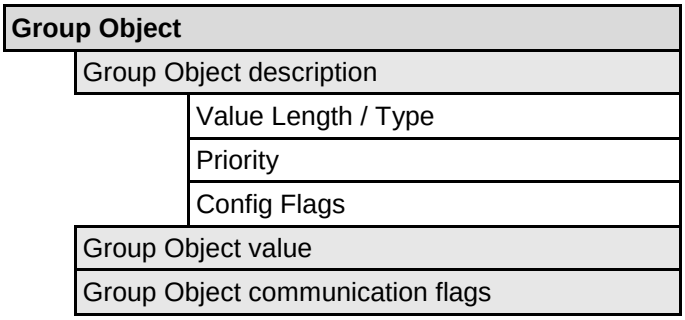
3.1 Overview

Group Objects may be distributed to a number of devices. Each device may be transmitter and receiver for Group Object values. More than one Group Object may exist in an end device and a Group Object in an end device may be assigned to one or more Group Addresses. Group Objects of an end device may belong to the same or to different groups. Each group has a network wide unique Group Address. The Group Address shall be mapped to a local group-index (TSAP) that shall be unique for the communication services of the end device by the Transport Layer. The Application Layer shall map the group-index by the Group Object Association Table to the group reference ID (ASAP) that shall access the Group Objects.

3.2 General data structure of the Group Object Table

In the sense of the previous clause a Group Object shall consist of three parts:

1. the Group Object description,
2. the Group Object value and
3. the Group Object communication flags.



The Group Object description must at least include the Group Object Type and the transmission priority.

The Config Flags shall include static information about the Group Object:

1. Read Enable
2. Write Enable
3. Transmit Enable
4. Communication Enable
5. Update Enable

The priority is urgent, normal or low.

The following Group Object Types are defined.

Group Object Type	Size of the Group Object Value
Unsigned Integer (1)	1 bit
Unsigned Integer (2)	2 bits
Unsigned Integer (3)	3 bits
Unsigned Integer (4)	4 bits
Unsigned Integer (5)	5 bits
Unsigned Integer (6)	6 bits
Unsigned Integer (7)	7 bits
Unsigned Integer (8)	1 octet
Unsigned Integer (16)	2 octets
Octet (3)	3 octets
Octet (4)	4 octets
Octet (6)	6 octets
Octet (8)	8 octets
Octet (10)	10 octets
Octet (14)	14 octets
Interface Object Reference	4 octets to 14 octets

Figure 2 - Group Object Types

Only Group Objects of the same Group Object Type shall be linked to the same Group Address and for Interface Objects references also the Interface Object Type with the same instance number must be the same.

The interpretation of the Group Object Value (data) is specified in [07].

The communication flags shall show the state of a Group Object. Following states shall be possible:

1. update
2. read_request
3. write_request
4. transmitting
5. ok, error

3.3 Group Object value transfers

3.3.1 Functionality

The application process shall trigger Group Object value transfers by "setting" or "clearing" the relevant communication flags of a Group Object. The Group Objects, or their images, shall be held in the Group Object Server. The communication flags shall play an essential role in triggering the Application Interface Layer's Group Object service to initiate the transfers. The local access to a Group Object of the Group Object server shall stimulate the Group Object server to initiate a network wide update of that Group Object. Complementary, if an update has been received, the local application shall be triggered to use the new value. There are four cases to be considered:

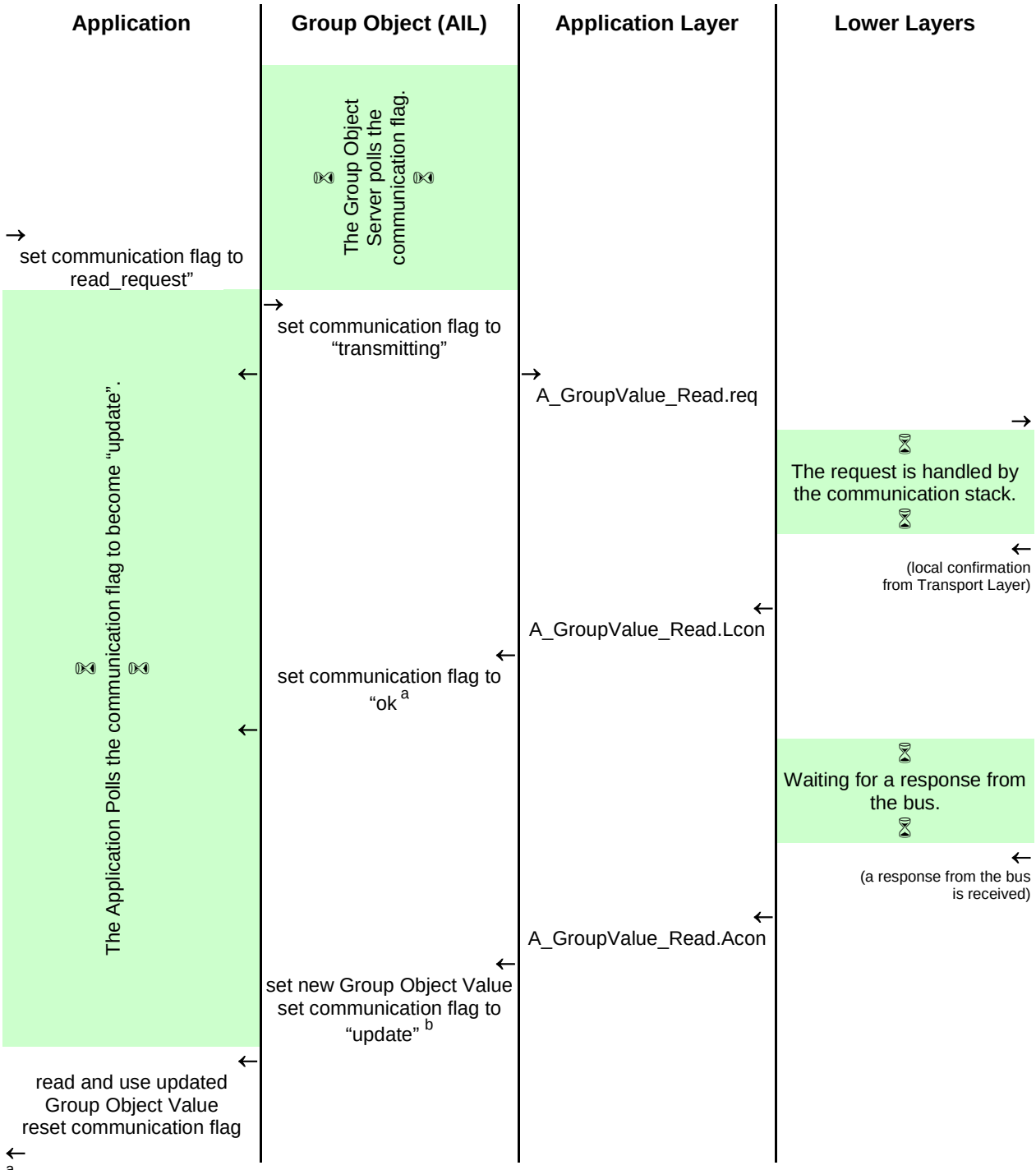
- the application wants
    1. to read the Group Object's value
    2. to write the Group Object's value or
- the Group Object service has received from the application layer
    3. a request to read the Group Object’s value
    4. an update on the Group Object's value.

The interaction between the application and the Group Object server are equivalent to the service primitives for request and indication, say it that in this case the exchange is subject to the status of the communication flags. This enables to distinguish local access from network wide update.

It is the responsibility of the internal or external user application program to trigger Group Object value transmissions.

3.3.2 Reading the Group Object Value

The following diagram gives the process flow when an application reads a Group Object value:



a If a negative A_GroupValue_Read.Lcon is received then the communication status shall be set to "error"; the actions following will not happen (no new value, no "update" flag).

b Update is subject to the "Update Enable" and "Communication" flag when used, else to the "Write Enable" flag.

Figure 3 - Reading a Group Object Value

3.3.3 Receiving a Request to Read the Group Object Value

The following diagram sketches the process flow when the Group Object Server receives a request to read a Group Object Value.

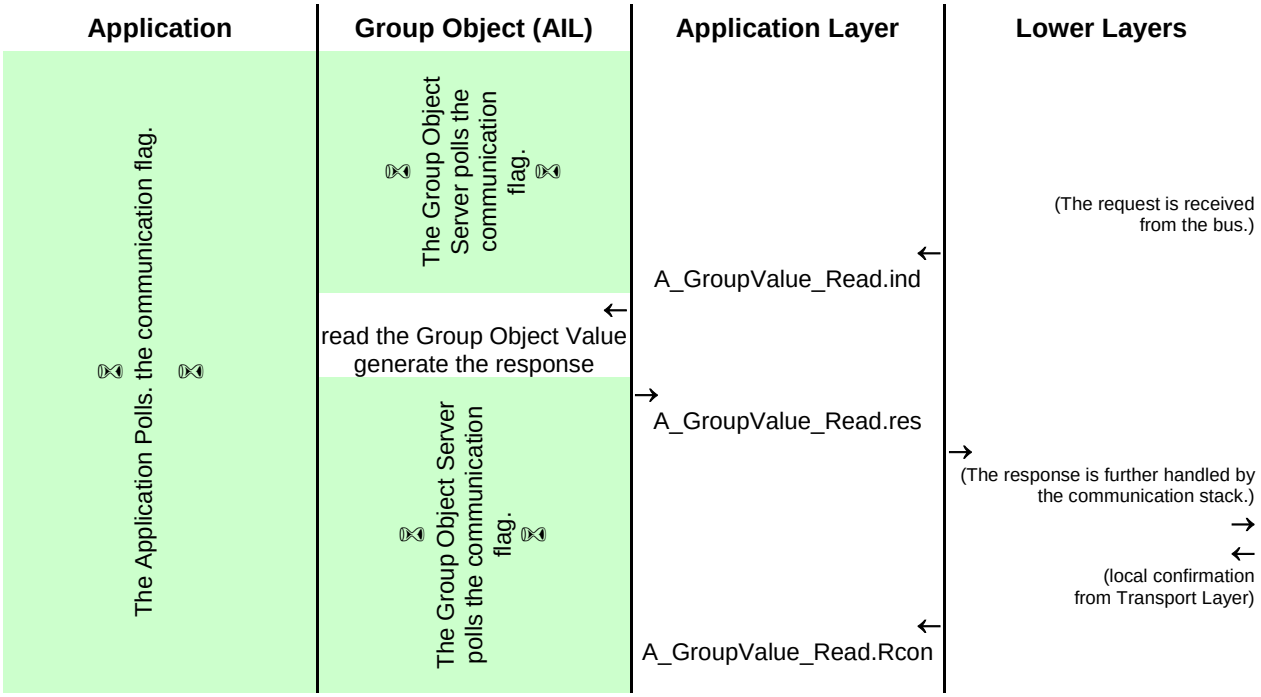
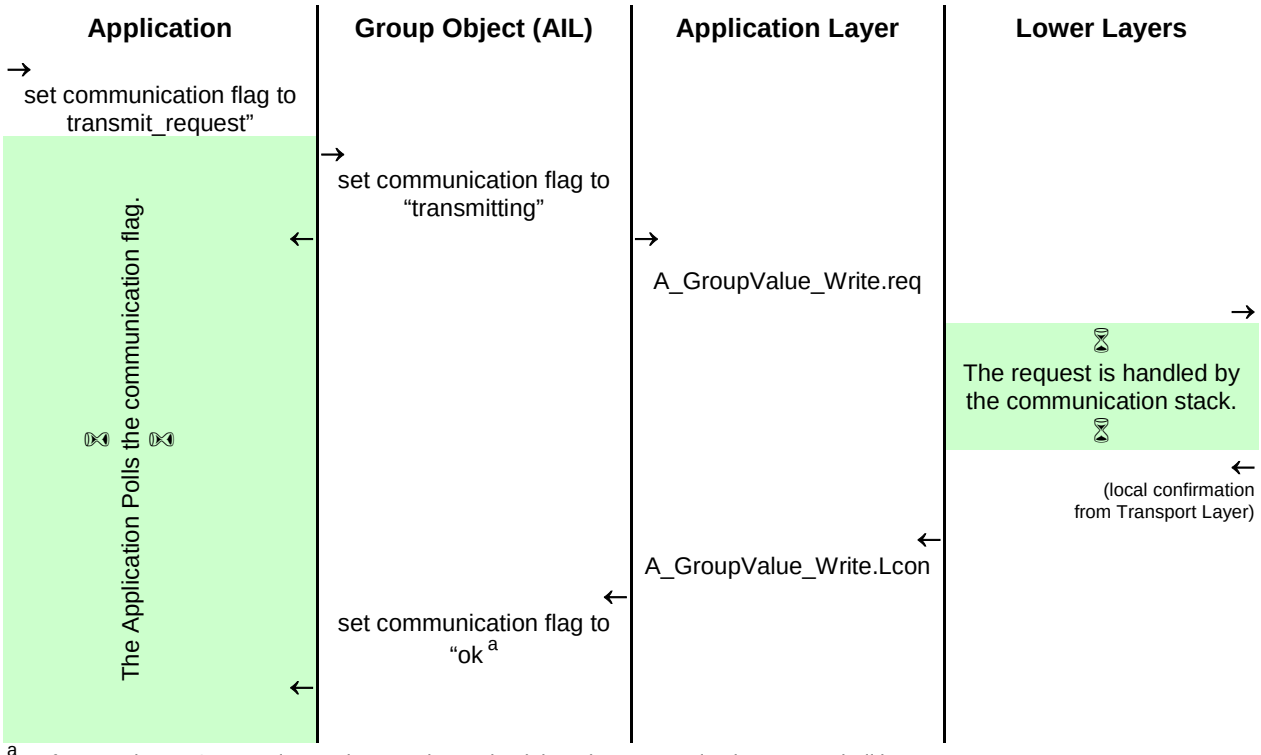


Figure 4 - Receiving a Request to read the Group Object Value

3.3.4 Writing the Group Object Value

The following diagram gives the process flow when an application writes a new value to a Group Object.

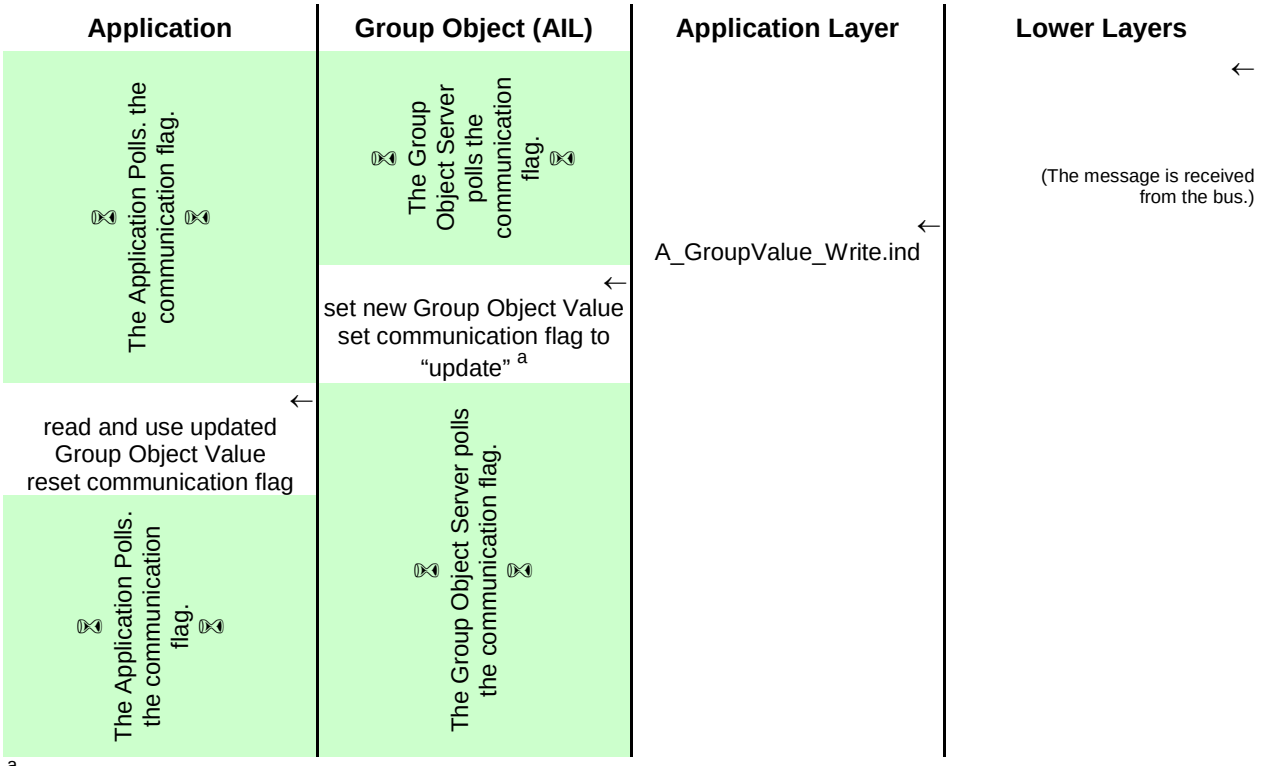


a If a negative A_GroupValue_Write.Lcon is received then the communication status shall be set to "error".

Figure 5 - Writing a Group Object Value

3.3.5 Receiving an Update on the Group Object Value

The following diagram sketches the process flow when the Group Object service receives a request to write a new value to the Group Object.



a This handling of the communication flag and the updating of the Group Object Value shall be subject to the configuration flag "write enable".

Figure 6 - Receiving an update of the Group Object Value

3.4 Group Object indirection – Group Object Handles and PID_OBJECT_VALUE (PID = 62)

Group Object Indirection denotes the requirement on the Application Interface Layer for certain Device Profiles to – next to the above specified mechanisms - provide an additional, parallel access to the values of the Group Objects in point-to-point connectionless or connection-oriented communication mode using the Function Property PID_OBJECT_VALUE,

In this, a Group Object shall be addressed in the device by a two octet Group Object Handle.

Please refer to the specification of PID_OBJECT_VALUE in [04].

4 Interface Object Server

4.1 Common structure

Interface Objects shall be instances of a common general structure. Interface Objects can be located either in the internal or in the external user application. Each Interface Object instance in a device shall have a unique identifier in the device, the object_index.

Each Interface Object in a device shall be addressed by an object_index. The object_index shall be unique within the device. Each Property of an Interface Object shall be addressed with a property_id. The property_id shall be unique for the Interface Object. For the A_PropertyDescription_Read, a Property may be addressed also by the property index. Each Interface Object shall consist of at least the Property Object Type. Interface Objects with active access protection are only accessible over point-to-point connection oriented communication relation ship.

All Interface Object shall base on the common structure as presented in Figure 7. Depending on the flavour of the Interface Object, part of this structure may not be present.

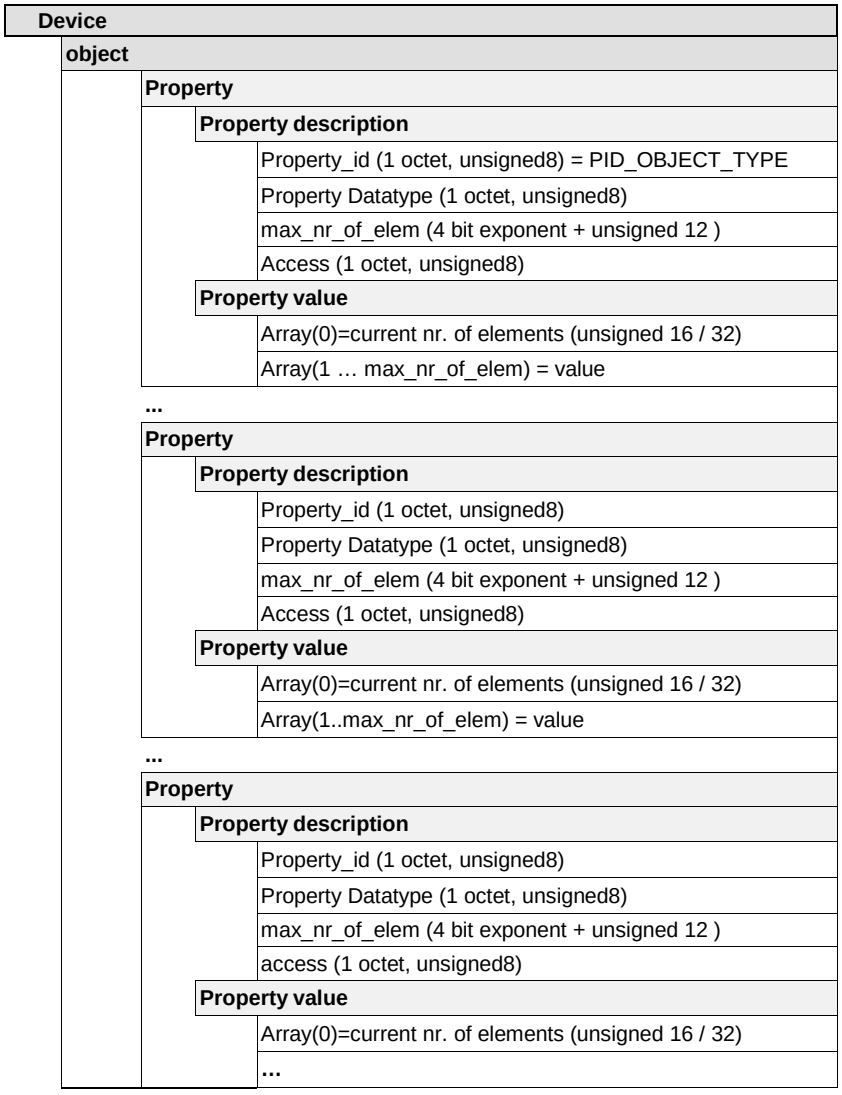


Figure 7 - Interface Object structure

From this common structure of Interface Objects, two types are derived:

1. Full Interface Objects, and
2. Reduced Interface Objects.

These types are specified in the following clauses.

NOTE If in the following specification the specific type is not explicitly mentioned or clear from the scope, then the requirements apply for both Full Interface Objects as well as for Reduced Interface Objects.

In a single device both types of Interface Objects (full and reduced) are able to co-exist. This means that the following cases are possible:

1. A device contains only Reduced Interface Objects.
2. A device contains both Reduced - and Full Interface Objects. This means that the service A_PropertyDescription_Read must be implemented in this device, but is only available for the Full Interface Objects.
3. A device contains only Full Interface Objects.

In [10] it is documented which combinations of Full - and Reduced Interface Objects are allowed.

4.2 Minimal requirements for Interface Objects

If any Interface Object is implemented in a device, then the Device Object shall be implemented. This Device Object shall have object_index 0.

In any Interface Object, the Property PID_OBJECT_TYPE is mandatory.

These rules apply both for mandatory Interface Objects as well as for optional Interface Objects (see below).

4.3 Types of Interface Objects

4.3.1 Full Interface Object

4.3.1.1 Definition

Full Interface Objects shall comply in full with the common data structure as specified in Figure 7.

Every Interface Object shall consist of a number of Properties. Every Property shall consist of

- one Property Description and
- one Property Value.

4.3.1.2 Property Description

The Property Description shall consist of

- the Property Identifier, and
- the Property Index, and
- the Write Enable flag, and
- the Property Datatype, and
- the Maximum Number of Elements information
- the definition of the read- and write access levels.

These elements are specified in the following paragraphs.

Property Identifier

The value of the Property Identifier shall be encoded in the field *property_id* of the A_Property-Description_Response-PDU.

Property Index

The Property Index of a Property shall be a unique number of for each within an Interface Object.

The first Property shall have Property Index 0 and further Properties shall be numbered with subsequent Property Indexes without gaps in the numbering.

The Property Index is used to read the Property Description in the A_PropertyDescription_Read-service.

Write Enable

The Write Enable flag shall specify whether the Property Value can be written through an A_PropertyValue_Write-service or not. The encoding shall be as follows:

0: The Property Value shall not be writeable, this is, the Property shall be read-only.

1: The Property Value shall be write-enabled.

Relation to access level: If this flag is cleared, the Property can not be written, independent of the set/used access level. If this flag is set, the access rights determine whether the Property can be written.

The value of the Write Enable flag shall be encoded in the field *w* of the A_PropertyDescription_-Response-PDU.

Property Datatype

The Property Datatype shall describe the data type of the Property. These Property Datatypes are specified in [09]. Every full Interface Object shall provide the most appropriate PDT for its Interface Object Properties, according the “Usage rules” specified there.

The value of the Property Datatype shall be encoded in the field *type* of the A_PropertyDescription_-Response-PDU.

Maximum Number of Elements

The value of a Property shall always be an array, as specified below. The Maximal Number of Elements shall specify the maximal number of elements of that array.

The value for max_no_of_elem shall be an unsigned 12 bit integer value with a 4 bit binary exponent without sign (the 4 most significant bits shall be the exponent).

The value of the Maximum Number of Elements shall be encoded in the field *max_nr_of_elem* of the A_PropertyDescription_Response-PDU.

Read - and write access levels

The attribute Access in the Property description shall indicate the necessary access level to read or write to the Property Value.

To obtain access to a protected Property Value, the Management Client shall apply the Management Procedure DM_Authorize2_RCo. This Management Procedure shall mainly be executed by the S-Mode Management Client when accessing devices of the Profiles System 2 and BIM M112 when the ETS®-user has provided an access key. This Management Procedure shall however also be followed by other Management Clients that support authorization. It is necessary when factory-released products need in a subsequent DM_SetKey to be locked by a key.

Table 1 lists the recommended access levels in function of the audience (management client).

Table 1 – Use of access levels for different purposes and Profiles

purpose	audience	access level	
		4 levels	16 levels
read access	runtime	level 3	level 15
end-user adjustable parameters	controller	level 3	level 3
configuration	ETS	level 2	level 2
product manufacturer	DevEdit/TransApp	level 1	level 1
system manufacturer	DevEdit/TransApp	level 0	level 0

4.3.1.3 Property Value

The value of a Property shall be an array with array index 1 to max_nr_of_elem. The array element ´0´ shall contain the current number (unsigned16 if max_no_of_elem exponent is 0 else unsigned 32) of valid array elements. The array can be reset to no elements by writing zero on element ´0´. The array shall automatically be extended if an element is written beyond the currently last element, but within the maximum allowed number of entries.

The Property with property_id = 1 and index 0 is named the Interface Object Type (PID_OBJECT_TYPE) and shall contain the description of the Interface Object itself. This Property is mandatory for every Interface Object.

Array of Interface Objects

If the maximum number of elements of the Property Interface Object Type is greater than 1 the whole Interface Object shall be an array of this Interface Object. Each Property of the array Interface Object must have at least the number of elements or a multiple of the number of elements of the Property Interface Object Type (PID_OBJECT_TYPE).

4.3.2 Reduced Interface Object

4.3.2.1 Structure

A Reduced Interface Object shall only support a subset of the common data structure Interface Objects, as depicted in Figure 8.

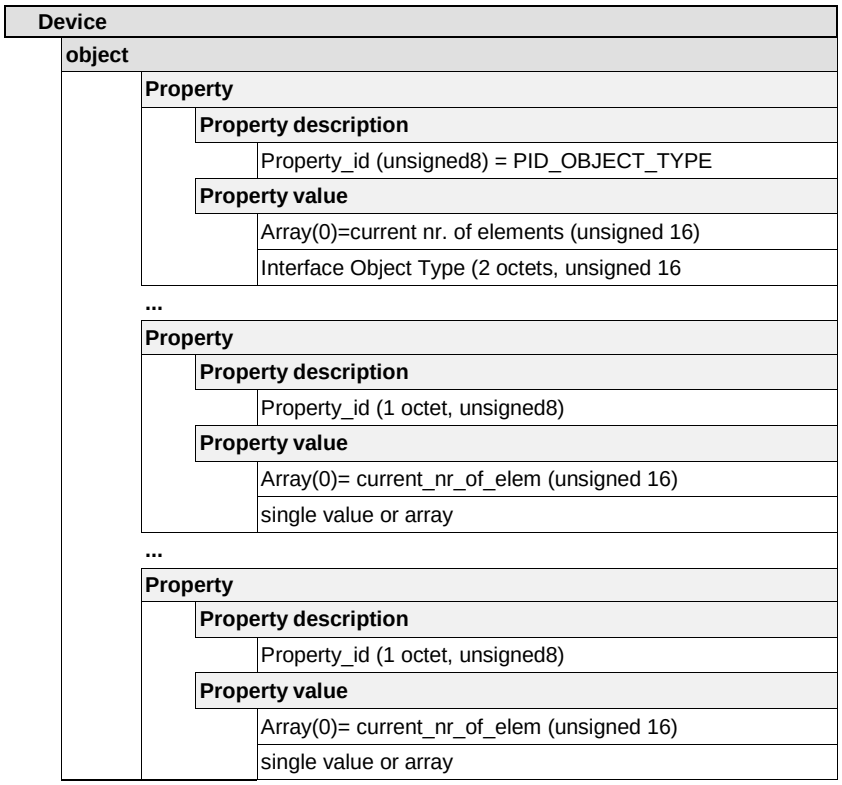


Figure 8 – Reduced Interface Object structure

This structure indicates that the Property Description shall be reduced to the Property_id. This Property_id shall not be readable by the A_PropertyDescription_Read-service, which is not supported for Reduced Interface Objects. The Property_id shall only be used for accessing the Property using the services A_PropertyValue_Read and A_PropertyValue_Write.

The support of the Reduced Interface Object requires the server side support of following APDUs:

A_PropertyValue_Read-PDU	(object_index [1 octet], property_identifier [1 octet], number_of_elements [4 bits], start_index [12 bits])
A_PropertyValue_Response-PDU	(object_index [1 octet], property_identifier [1 octet], number_of_elements [4 bits], start_index [12 bits] data [up to 10 octets])
A_PropertyValue_Write-PDU	(object_index [1 octet], property_identifier [1 octet], number_of_elements [4 bits], start_index [12 bits] data [up to 10 octets])

The behaviour of these Application Layer services is unchanged compared to the specification in [02].

The A_PropertyDescription_Read-service shall not be supported for a Reduced Interface Object. If a device receives an A_PropertyDescription_Read-PDU with the object_index referring to a Reduced Interface Object then the device shall show no reaction, this is, no A_PropertyValue_Response-PDU is generated.

4.3.2.2 Property Value is a single Value

The Property Value shall be accessed by the services A_PropertyValue_Read and A_Property-Value_Write. These services shall allow the handling of arrays, which means, that the PDUs shall always contain the fields nr_of_elem and start_index. In case of a single value, this shall be accessed by setting these fields to nr_of_elem = 1 and start_index = 1. A start_index = 0 shall refer to the current_nr_of_-elements, which shall be 1 in case of a single value. Therefore an A_PropertyValue_Read.req with nr_of_elem = 1, start_index = 0 shall result in an A_PropertyValue_Read.res(nr_of_elem = 1, start_index = 0, data = 1) in this case.

4.3.2.3 Property Value is an Array

In case the Property value contains an array, this shall have the following structure:

Array(0) = current_nr_of_elem (2 octets, unsigned 16)
Array(1 … max_nr_of_elem) = value

This means that element zero of the array shall always contain the current number_of_elem, stored in two octets (independent of the datatype of the array). Elements 1 and further shall contain the array elements.

NOTE The datatype and the max_nr_of_elements must be known by the client as this information cannot be retrieved for Reduced Interface Objects (the A_PropertyDescription_Read-service is not available).

4.3.2.4 Comparison to full Interface Objects

In comparison with Full Interface Objects the following information shall not be available due the lack of the A_PropertyDescription_Read-service:

- Property Datatype (PDT),
- access levels,
- maximum no. of elements in case of an array,
- access to Properties by the property_index

  In case of Full Interface Objects an A_PropertyDescription_Read can use the property_index to obtain the property_id. This is not possible for Reduced Interface Objects.

This means that a Management Client cannot retrieve this information from the device, but must have implicit knowledge about:

- which Properties exist in a Reduced Interface Object, and
- which is the Property Datatype (PDT) of each Property, and
- whether or not the Property contains a single value or an array, and
- the maximum no. of elements in case the Property value is an array what is, and
- what are the access levels.

  Properties of Reduced Interface Objects shall be accessible without prior authorisation. Reading and writing the Property value shall be possible with the lowest access level.

4.3.3 Error handling

For the services A_PropertyValue_Read and the A_PropertyValue_Write the same error handling shall apply as for full Interface Objects. This means that if an A_PropertyValue_Read-PDU or an A_PropertyValue_Write-PDU contains a field with an invalid value, the device shall answer with an A_PropertyValue_Response-PDU containing the same parameters, but an empty data field.

Please refer to the full error handling of these Application Layer services in Chapter 3/3/7 “Application Layer”.

This behaviour shall apply in the following cases.

- The Interface Object with requested object_index does not exist in the device [invalid object_index].
- In the requested Interface Object there is no Property with the requested Property_id [invalid property_id].
- The field nr_of_elem contains zero or a number that corresponds to an invalid nr_of_elements (i.e. >1 in the case of a single value or greater than the current_nr_of_elem above the start_index in case of an array) [invalid element count].
- The start_index is >1 in case of a single value or greater than the current_nr_of_elem in case of an array [invalid start_index].

4.4 Types of Properties

4.4.1 Data Properties

The Properties and the Property Description of Data Properties are typically accessed via the Application Layer services for Properties on a point-to-point connectionless - or connection oriented communication mode.

♦ **Services**

- A_PropertyValue_Read
- A_PropertyValue_Write
- A_PropertyDescription_Read

4.4.2 Function Properties

A Function Property shall allow a communication partner to call a function in the device. Function Properties shall be typed by the Property Datatype PDT_Function (value = 3Eh)..(Property Datatypes are specified in [09].)

♦ **Services**

The support of Function Properties requires the server side support of the following services.

- A_FunctionPropertyCommand-service

  This requires the support of the following APDUs:
  - A_FunctionPropertyCommand-APDU (object_index, property_id, data)
  - A_FunctionPropertyState_Response-APDU(object_index, property_id, return_code, data)
- A_FunctionPropertyState_Read-service

  This requires the support of the following APDUs:
  - A_FunctionPropertyState_Read-APDU (object_index, property_id, data)
  - A_FunctionPropertyState_Response-APDU(object_index, property_id, return_code, data)

Function Property Services shall be implemented and used as an entire “set”. This is, the services A_FunctionPropertyCommand (1 service primitive) and A_FunctionPropertyState_Read (2 primitives) shall always be implemented together.

A Function Property shall be a Property within the same object model as a Data Property. A Function Property can be implemented within Full Interface Objects or Reduced Interface Objects. For Full Interface Objects the specific A_PropertyDescription_Read-service is as specified in [02].

A Function Property differs from the above specified Data Property by the fact that the data transmitted to the Property by the Management Client *may* differ in format from the data responded by the Management Server as a response.

The format of data in request and response service primitives depends on the specific Function Property and is specified in [04].

A common maximum reaction time of 1 second shall be maintained for Function Properties.

General guideline

If the data in the request, response and indication service primitives is

- the same, then the services
  - A_PropertyValue_Read
  - A_PropertyValue_Write

  should be used;

- different, then the services
  - A_FunctionPropertyCommand
  - A_FunctionPropertyState_Read

  should be used.

In general, Function Properties shall not be used during runtime. Exceptions shall be included in the Function Property definition. PID_OBJECT_VALUE (PID = 62) as specified in [04] may be used during runtime.

4.4.3 Network Parameter Properties

♦ **Services**

- A_NetworkParameter_Read
- A_NetworkParameter_Write

4.5 Ways for addressing Interface Objects

There are 2 different ways to address Properties of an Interface Object in a device.

1. → IndividualAddress → Interface Object Index →Property ID or
2. → Group Address → Object Instance →Property ID

To access the Property Description there are also two different ways:

1. → IndividualAddress → Interface Object Index → Property ID or
2. → IndividualAddress → Interface Object Index → Property Index

4.6 Interface Object Interworking

Standards for Interface Objects have been set for indication of the Property Datapoint Types, Property Identifiers and the Object Types. Please refer to [08] and [11].

4.7 System Interface Objects

System Interface Objects are Interface Objects designed for network - and application management. No System Interface Objects are mandatory for a device. These are the 4 most important System Interface Objects:

1. the Device Object
2. the Address Table object
3. the Association Table object
4. the Application Program object.

The purpose of System Interface Objects is to enable uniform Network Management. This shall support the end-user during the configuration of end devices through predefined System Interface Objects that offer the necessary Properties. See [03] for more details.

4.8 Defining Application Interface Objects

4.8.1 General

Standard Application Interface Objects are defined in the various application descriptions in the Chapters of Volume 7 ([11]).

It is possible to create proprietary Application Interface Objects. These shall be created according to the structures and datatypes given in clause 4.1 “Common structure” of this document.

Application Interface Objects may also be accessed by the Group Object Server via a reference in the Group Object Table.

4.8.2 Interface Object Server for own Application Interface Objects

Application Interface Objects have the same structure as System Interface Objects and therefore they use the same Interface Object Server.

4.8.3 Interface Object Client for Accessing Remote Application Interface Objects

An Interface Object Client can be external or internal.

An internal Interface Object Client may be based on the internal message format of A_Property-Description_Read, A_PropertyValue_Read and A_PropertyValue_Write request and confirmation messages.

The interface of the Interface Object Client at the external user application depends on the EMI chosen by the LM_Switch message, see [06] clause "Layer Access Management".

4.8.4 Message flow for Property Services

4.8.4.1 Scope

In the following paragraphs the message flow for Property handling is described in a client/server model.

4.8.4.2 Property Value Read

Interface Object Client		Interface Object Server
A_PropertyValue_Read.req →		
A_PropertyValue_Read.Lcon ←		→ A_PropertyValue_Read.ind
		← A_PropertyValue_Read.res
A_PropertyValue_Read.Acon ←		→ A_PropertyValue_Read.Rcon

The Application Layer User in the client shall use the A_PropertyValue_Read.req service-primitive to read the value of a Property of an Interface Object in the Interface Object Server. The Interface Object Server shall be addressed with a local ASAP that shall be mapped to an Individual Address by the Transport Layer in the Interface Object Client. The Interface Object in the Interface Object Server shall be addressed with an object_index and the Property of the Interface Object shall be addressed with a property_id. The nr_of_elem and start_index shall indicate the number of array elements starting with the given start_index in the Property Value that the user wants to read. The service shall be confirmed by the Application Layer in the Interface Object Client with an A_PropertyValue_Read.Lcon.

The user of Application Layer in the Interface Object Server shall receive an A_PropertyValue_Read.ind and shall respond with an A_PropertyValue_Read.res. The Transport Layer in the Interface Object Server shall transmit the A_PropertyValue_Read.res with a TSDU = A_PropertyValue_Response-PDU. The A_PropertyValue_Read.res service primitive shall be locally confirmed in the Interface Object Server with an A_PropertyValue_Read.Rcon.

The Application Layer User in the Interface Object Client shall receive an A_PropertyValue_Read.Acon with the requested data.

If the Interface Object Server has a problem to respond to the A_PropertyValue_Read.ind, e.g. Interface Object or Property does not exists or the requester has not the required access rights the nr_of_elements in the A_PropertyValue_Response-PDU shall be set to zero and shall contain no data.

4.8.4.3 Property Value Write

Interface Object Client		Interface Object Server
A_PropertyValue_Write.req →		
A_PropertyValue_Write.Lcon ←		→ A_PropertyValue_Write.ind
		← A_PropertyValue_Read.res
A_PropertyValue_Write.Acon ←		→ A_PropertyValue_Write.Rcon

The Application Layer User in the Interface Object Client shall use the A_PropertyValue_Write.req service primitive to write the value of a Property of an Interface Object in the Interface Object Server. The Interface Object Server shall be addressed with a local ASAP that shall be mapped to an Individual Address by the Transport Layer in the Interface Object Client. The Interface Object in the Interface Object Server shall be addressed with an object_index and the Property of the Interface Object shall be addressed with a property_id. The nr_of_elem and start_index shall indicate the number of array elements starting with the given start_index in the Property Value that the Interface Object Client wants to write. The service shall be confirmed by the Application Layer in the Interface Object Client with an A_PropertyValue_Write.Lcon.

The Application Layer User in the Interface Object Server shall receive an A_PropertyValue_Write.ind and shall respond with an A_PropertyValue_Read.res. The Transport Layer in the Interface Object Server shall transmit the A_PropertyValue_Read.res with a TSDU = A_PropertyValue_Response-PDU. The A_PropertyValue_Read.res service primitive shall be locally confirmed in the Interface Object Server with an A_PropertyValue_Write.Rcon.

The Application Layer User in the Interface Object Client shall receive an A_PropertyValue_Write.Acon with the written data.

4.8.4.4 Property Description Read

Interface Object Client	Interface Object Server
A_PropertyDescription_Read.req →	
A_PropertyDescription_Read.Lcon ←	→ A_PropertyDescription_Read.ind
	← A_PropertyDescription_Read.res
A_PropertyDescription_Read.Acon ←	→ A_PropertyDescription_Read.Rcon

The Application Layer User in the Interface Object Client shall use the A_PropertyDescription_Read.req service to read the value of a Property of an Interface Object in the Interface Object Server. The Interface Object Server shall be addressed with a local ASAP that shall be mapped to an Individual Address by the Transport Layer in the Interface Object Client. The Interface Object in the Interface Object Server shall be addressed with an object_index and the Property of the Interface Object shall be addressed with a property_id. The service shall be confirmed by the Application Layer in the Interface Object Client with an A_PropertyDescription_Read.Lcon.

The Application Layer User in the Interface Object Server shall receive an A_PropertyDescription_-Read.ind and respond with an A_PropertyDescription_Read.res. The Transport Layer in the Interface Object Server shall transmit the A_PropertyDescription_Read.res with a TSDU = A_Property-Description_Response-PDU. The A_PropertyDescription_Read.res service primitive shall be confirmed locally in the Interface Object Server with an A_PropertyDescription_Read.Rcon.

The Application Layer User in the Interface Object Client shall receive an A_PropertyDescription_-Read.Acon with the requested data.

5 File Server

5.1 File Server model

The File Server shall be coupled with the internal file system of the device and it shall communicate with the Application Layer via Application Layer services. An application only communicates with the internal file system.

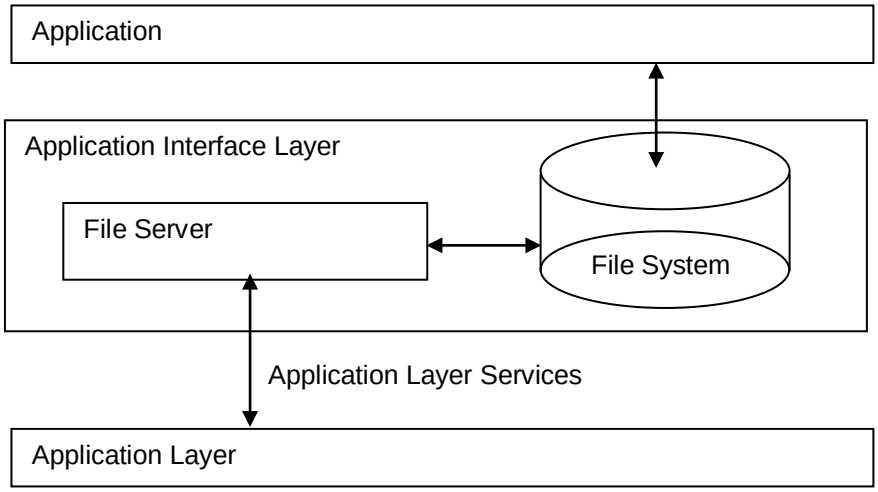


Figure 9 – Basic File Server model

In a closer look, the File Server itself consists of several independent blocks, which communicate with each other.

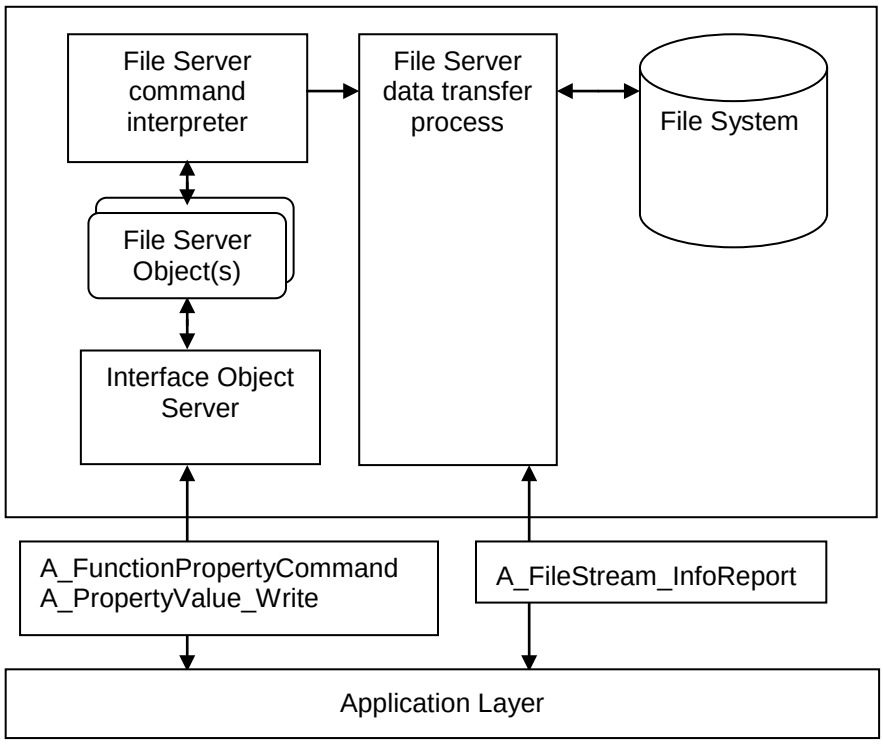


Figure 10 – Blocks of the File Server

The basic function of the File Server is to give a File Client access to the file system of the server. The client has to tell the server which file is of interest and what has to be done with it. For this communication, an Interface Object of type File Server Object shall be used. Properties of this Interface Object shall be used to structure this communication.

The Property PID_FILE_PATH shall contain the path name of the file to which the requested operations, e.g. file transfer request, apply.

Via the Property PID_FILE_COMMAND, a command for the file specified by the path name can be given. The command is always sent after the path name has been set.

It is also possible to include the path name into the command datagram. This is very useful, if long frames are used. For short frames the path name can only have a maximum length of 11 octets if it is transferred with the command. In a short frame environment, it is recommended, that the Property PID_FILE_PATH is always used for the transmission of a path name. If a path name is transferred with a command, it will override the path name stored in the Property PID_FILE_PATH.

To allow simultaneous access to several files from different clients (or from the same client) the concept of a File Handle shall be used. Before a client writes to the File Server Object, it retrieves a File Handle from this File Server Object. Once having a valid handle, the File Server Object is exclusively reserved for this client. All other clients will be blocked by getting an invalid file handle telling that this File Server Object is occupied.

When a File Server Object returns a valid File Handle to a File Client, it stores the Individual Address of this client. Further commands are only accepted from this client.

The reservation of a File Handle for a specific client shall be released either after the execution of a file command or after a time-out of approx. 6 s, if no command is executed i.e. the client does not communicate with this File Server Object.

A File Server Object shall only handle one file at a time. A File Server that can handle more than one File at a time shall implement as many instances of the File Server Object as files that it can handle simultaneously.

5.2 File Formats

5.2.1 General

File data shall always be stored and retrieved in binary form. During transmission no changes of file data are performed when using an FTP command.

By convention, text files use the PC standard CR/LF (0Dh, 0Ah) line termination.

5.2.2 Short HTML File Format

5.2.2.1 Format

Files containing HTML code are usually displayed in a browser window. It is convenient for the end user if the retrieval of HTML files is done in high speed. To achieve this, only a part of the whole file is transferred. If the “Get File” command is used, and the returned Content-Type is “text/short-html-xxx”, then the server expects the client to add a heading part and a footing part automatically.

```
<html>
    <head>
…
        <table align="center" border="0" width="640" bgcolor="#BFCFDF">
            <tr>
```

Figure 11 – Heading Part for a short HTML File

```
                    <td align="center">
...
                    </td>
```

Figure 12 – Main Part of a short HTML File

```
                </tr>
            </table>
…
            </body>
</html>
```

Figure 13 – Footing Part of a short HTML File

The transferred main part of the HTML file is only the contents of a predefined HTML table. The table definition is given in the heading part. The width of the table (640 in the example above) is defined by the returned Content Type (here: text/short-html-640).

5.2.2.2 Example

The example below shows a real HTML file with heading and footing part. It can be seen that the reduction in transfer is quite high.

```
<html><head>
<meta http-equiv="cache-control" content="no-cache">
</head>
<body bgcolor="#9AB3CD">
<center> </center>
<table align="center" border="0" width="690" bgcolor="#BFCFDF">
<tr><td align="center" valign="middle"><h2>
1.2.3
</h2></td></tr></table>
<center> </center>
<table align="center" border="0" width="640" bgcolor="#BFCFDF"><tr>
<!—- end of heading part -->

<td align="center">
<a href="cfga">[  A  ]</a> <a href="cfgb">[  B  ]</a>
<a href="cfgc">[  C  ]</a> <a href="cfgd">[  D  ]</a><br>
<a href="/1.2.3/en/cha">[home]</a>
<a href="/1.2.3/en/obj/obja">[objects]</a>
<p><b>  A  </b><p>
<form method="GET">
<input type="text" value="  A  " name="NA" size="11"><p>
<input type="submit" value="SET NAME (11)" name="NA">
</form><p>
(SET NAME stops application for 100ms)
<p></td>

<!—begin of footing part -->
</tr></table>
<center> </center>
```

```
<table align="center" border="0" width="690" height="30" cellpadding="3"
cellspacing="0"><tr>
<td align="center" bgcolor="#BFCFDF">
<h2><A HREF=".">zur&#252;ck</A></h2>
</td></tr></table>
<p align="center">&copy; 2006 Lingg &amp; Janke
</p></body></html>
```

5.2.3 Directory Listing Format

5.2.3.1 Introduction (informative)

Directory listings are dedicated files with a defined content. Two directory listing formats are defined.

1. The short format with a limited file size of 65 535 octets (16 bit length) and no date and time stamp. File names may have up to 255 characters.
2. The long format with a file size of up to 4 Gigabyte (32 bit length) and with date and time stamp. File names also may have up to 255 characters.

The format of the directory listing is the major limitation for the file system of the File Server. Limits are given for the maximum file size and the name length of file names and directory names.

For most small KNX devices it is sufficient to have a file size of 64 Kbyte. In a KNX TP1 bus environment, large files are very uncommon due to the long transfer times. And finally a clock is usually not provided to give files a date and time stamp.

For these devices, the short directory format is very suitable. Only the file size and the file name are transmitted. But, and this is the major advantage, directory entries with file names of up to 9 characters fit into one datagram of the A_FileStream_InfoReport service (if short frames are used). The transmission of a directory listing becomes very fast.

If the short directory format does not serve the needs of the KNX device, the long directory format shall be used.

The File Client shall be able to decode both directory formats. It is possible to distinguish both formats during the decoding process.

The transfer of a directory listing coded as described below is self optimizing and reduces busload to a minimum.

5.2.3.2 Short directory listing format

5.2.3.2.1 Transfer

Every file name or directory name in a short directory listing shall start with an A_FileStream_InfoReport datagram.

The whole directory listing shall be a sequence of A_FileStream_InfoReport datagrams.

The listing shall end if a A_FileStream_InfoReport datagram with data length 0 is sent.

5.2.3.2.2 Format

The first datagram shall contain an identifier, the file name length, the file size as an unsigned 16 bit integer and the file name or directory name. The identifier shall be used to distinguish whether the name is a file name or a directory name. And it also contains information about the type of the directory listing.

Limitations of the short directory listing format:

- max file size 64 Kbyte
- max name length for file names or directory names 255 characters
- no date and time

octet (in A_FileStream_InfoReport_PDU)	9	10	11	12	13 (MSB) to N
description	identifier	file name length	file size (binary)		file name / directory name (with short frames: first 9 characters)
		0 to 255	high	low	

Figure 14 – First Datagram of a short directory listing entry

octet (in A_FileStream_InfoReport_PDU)	9 (MSB) to 21
description	file name or directory name (with short frames: next 13 characters)

Figure 15 – Following Datagrams of a short directory listing entry (only for names longer than 10 characters if short frames are used)

♦ **Identifier**

The first octet shall be the identifier and shall be used for both directory listing formats.

bit	7 (msb)	6	5	4	3	2	1	0 (lsb)
description	0	0	0	0	0	0	LL	DN

Figure 16 – Identifier Octet

- DN (directory name)
  - 0 the following name shall be a file name
  - 1 the following name shall be a directory name
- LL (long listing)
  - 0 the following directory list shall be using the short directory listing format
  - 1 the following directory list shall be using the long directory listing format
- Unused bits shall be 0

Identifier	File name Length	Meaning
00h	09h	file name with 9 chars (fits into one datagram)
00h	23h	file name with 35 chars (two extra datagrams are needed if short frames are used)
01h	09h	dir name with 9 chars (fits into one datagram)
01h	16h	dir name with 22 chars (a second datagram is needed if short frames are used)

Figure 17 – Examples for the Identifier and File name Length octets in a Short Directory Listing

high	low	Meaning
00h	FFh	File size is 255 octets
04h	00h	File size is 1024 octets
80h	00h	File size is 32768 octets
FFh	FFh	File size is 65535 octets

Figure 18 – Examples for the unsigned 16 bit integer used for the file size

octet 13	octet 14	octet 15	octet 16	octet 17	octet 18	octet 19	octet 20	octet 21
61h	62h	63h	64h	65h	2Eh	74h	78h	74h
a	b	c	d	e	.	t	x	t

Figure 19 – Example of a file name that fits into the first short frame datagram. A leading "/" (slash) is not transmitted.

5.2.3.3 Long directory listing format

5.2.3.3.1 Transfer

Every file name or directory name in a long directory listing shall start with a new A_FileStream_InfoReport datagram.

The whole directory listing shall be a sequence of A_FileStream_InfoReport datagrams.

The listing shall end if a A_FileStream_InfoReport datagram with data length 0 is sent.

5.2.3.3.2 Format

The first datagram shall contain an identifier, the file name length, the file size encoded as a 32 bit unsigned integer, the file date (3 octets, DPT_Date, DPT_ID = 11.001), the file time (3 octets, DPT_TimeOfDay, DPT_ID = 10.001) and the filename or directory name. The identifier shall be used to distinguish whether the name is a filename or a directory name. It shall also contain information about the type of the directory listing.

Limitations of the long directory listing format:

- max file size 4 Gigabyte
- max name length for filenames or directory names 255 characters
- with date and time

octet (in A_FileStream_InfoReport_PDU)	9	10	11	12	13	14	15	16	17
description	identifier	file name length	file size (32bit unsigned integer)				Date (DPT_Date)		
		0 to 255	high	mhigh	mlow	low	Day	Month	Year

18	19	20	21 (MSB) to N
Time (DPT_TimeOfDay)			File name / directory name
Hour	Minutes	Seconds	

Figure 20 – First Datagram of a Long Directory Listing Entry

octet (in A_FileStream_InfoReport_PDU)	9 MSB to 21
description	file name or directory name (next 13 characters)

Figure 21 – Following datagrams of a long directory listing entry (only for names longer than 1 character if short frames are used)

The first octet shall be the identifier and shall be used for both directory listing formats.

bit	7 (msb)	6	5	4	3	2	1	0 (lsb)
description	0	0	0	0	0	0	LL	DN

Figure 22 – Identifier octet

- DN (directory name)
  - 0 the following name shall be a file name
  - 1 the following name shall be a directory name
- LL (long listing)
  - 0 the following directory list shall be using the short directory listing format
  - 1 the following directory list shall be using the long directory listing format
- Unused bits shall be 0

Identifier	File name Length	Meaning
02h	01h	file name with 1 character (fits into one datagram)
02h	1Bh	file name with 27 characters (two extra datagrams are needed if short frames are used)
03h	01h	directory name with 1 character (fits into one datagram)
03h	0Eh	directory name with 14 characters (a second datagram is needed if short frames are used)

Figure 23 – Examples for the Identifier and File name Length octets in a Long Directory Listing

high	mhigh	mlow	low	Meaning
00h	00h	00h	FFh	File size is 255 octets
00h	00h	FFh	FFh	File size is 65535 octets
80h	00h	00h	00h	File size is 2147483648 octets
FFh	FFh	FFh	FFh	File size is 4294967295 octets

Figure 24 – Examples for the unsigned 32 bit integer used for the file size

octet 21
61h
a

Figure 25 – Example of a file name that fits into the first short frame datagram. A leading "/" (slash) is not transmitted.