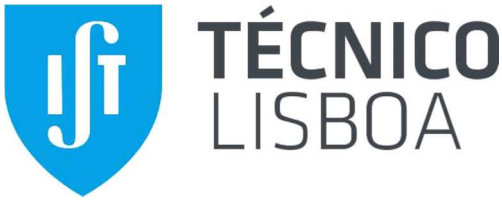


Master in Information Systems and Computer Engineering

Ambient Intelligence

2ND SEMESTER - 2015/2016

DomoBus Communications Library
Project Report

Group G06

Rafael Afonso Rodrigues – 85343

ABSTRACT

The current report details the complete implementation of the DomoBus Communications Library project, which entails the development of a communications module, written using the Java programming language, that satisfies the specifications, guidelines and protocols defined by the DomoBus technology, as it has yet to be carried out. It goes through the details of the proposed approach, evidencing the required and useful functionalities for such library, where three core components are defines: a network gateway, a handler for messages and an API. The resulting implementation, however, possess a structure with a not so strict separation and goes towards a centralized operations manager, which divides the workload onto smaller self-contained categories. Further in the report, an implementation of the library built is tested in a simulated environment as to evaluate the followed approach. And, while the results show the practically and operationally of the designed solution, some minor issues are apparent. Although it goes with the sentiment that the project detailed here needs to be extended and further tested as to reach a more concrete contribution.

Keywords: communications library, generic supervision framework

INDEX

1. Introduction

The project, presented through this report, aims to implement, following the DomoBus specification guidelines, a generic and malleable communications library for the Supervision Level of a DomoBus network, which is also referred to as DCOMM, and to create a test case by simulating multiple concurrent and interacting applications to determine the feasibility of the developed library for such networks.

Its objective is therefore to establish a customizable framework for Supervisor applications to communicate with each other. Not only abstracting the inner-workings of the network but also facilitating the integration of heterogeneous Supervisors, granting them the opportunity of providing different services and allowing complete control over how such services are processed internally. Furthermore, it also extends beyond the simple communications overlay and is designed to provide a basic management structure for the Supervision Level.

The core functionalities to be implemented are the following:

- Management of peer-to-peer connections and UDP datagrams exchanges across the network, with retransmission protocols.
- Creation, transmission and processing of DomoBus Supervision Level packets and data formats validation.
- Asynchronous and synchronous call methods for all available Control Level and Supervision Level commands through a standard interface.
- Local and remote error handling.
- Management of the notifications broadcasting and the Publisher-Subscriber system between the network's devices and applications.
- Basic identity control mechanisms and support for DomoBus Name Servers registration and querying.
- Continuous monitoring and audit with multi-event logging.

2. Related Work

This project center exclusively around the DomoBus automation technology and more precisely with the Supervision level and its interactions with the Control Level and itself within the DomoBus network.

2.1. DomoBus

DomoBus is a proprietary technology specification intended for home automation that is currently being developed. It sets a generic framework that allows for the control of physical devices and sensors across multiple rooms, or even system wide, and defines basic operations for their management **(1-4)**:

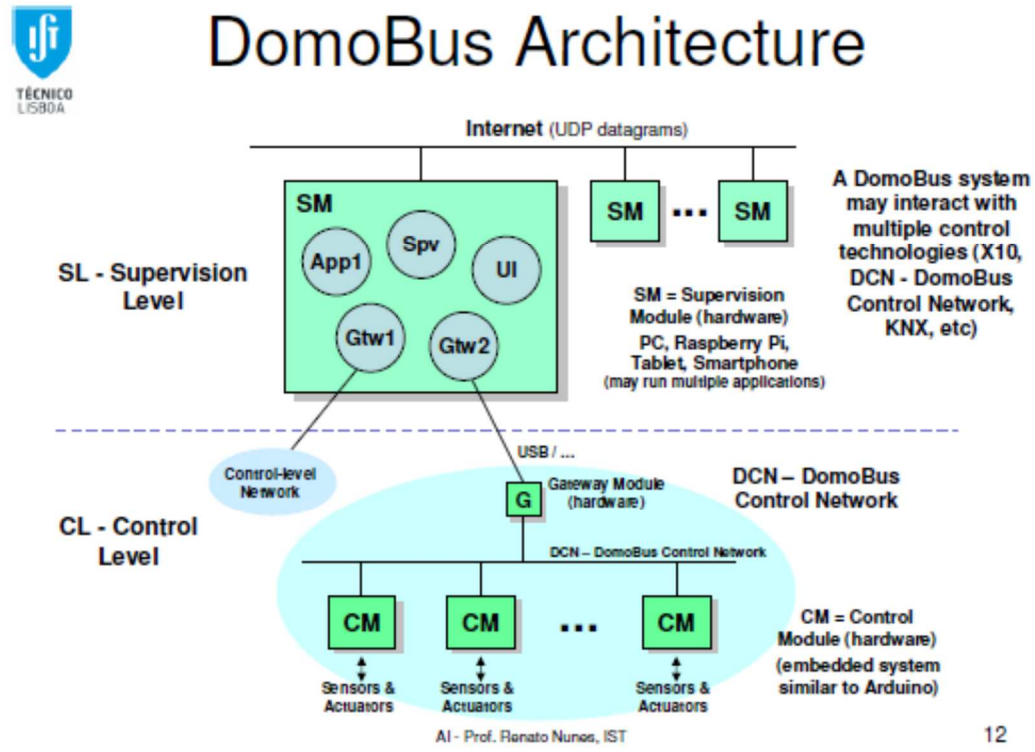
- GET: retrieve the current value of an element's property;
- SET: updates an element's property with the given value;
- NOTIFY: notifies an update of the value of a given element's property.

Each element is composed of multiple properties and follow the DomoBus specification language, which defines a generic XML-based data format they all use **(4)**.

At the low-level, physical devices and sensors are connected to each other through a control module. In turn it can itself be connected to other control modules and increase the networks size.

However, the DomoBus technology doesn't narrow itself as it goes beyond this to reach higher-level interactions and defines protocols and systems that can operate using TCP/IP networks **(1)**, effectively expanding its reach.

2.1.1. Network Architecture



A DomoBus network is composed of two levels: one said to be low-level, called Control Level; and another one that functions at a higher level, in the perspective of the network, referred as Supervision Level.

As stated before, the control modules are directly connected to physical devices and sensors, they are responsible for managing them and allowing external interactions through exchanged commands.

On the other hand, supervision modules, referred to Supervisors by onwards, provide them similar services, as they link the multiple Control Level networks, connected to them through gateway modules. Furthermore, they're also connected to other supervision modules and since they operate using TCP/IP networks, interactions by devices that are separated by huge distance are possible.

2.1.2. Data Formats

Its current documentation, **(1)**, specifies multiple data formats, especially for the low-level automation, but also for the packets being used at the higher level, or Supervision Level, which are the ones that will be used during the implementation of this project.

2.2. Development Environment

2.2.1. Programming Language

The programming language used is Java, more specifically the version 7 of its implementation.

Java is an object-oriented, generic and concurrent programming language, whose source code is organized in classes and compiled into package files called Java Archive, or JAR, allowing for the distribution of applications or libraries.
Those JAR files can be simply extracted when dealing with libraries, or executed, in the case of applications, using a Java Virtual Machine, or JVM **(2)**. Effectively rendering the applications developed multi-platform without any further modification and going in the desired direction of supporting heterogeneous Supervisors.
Although, precautions are needed since some differences can appear across the multiple operating systems supported by the JVM, especially when dealing with local file system paths and processes commands.
The JVM, and the language more generally, is currently maintained and owned by Oracle Corporation and distributed as an open-source software.

Even though the Java language isn't as low level as others like C or C++, it can achieve similar performances especially when dealing with base operations. And as other advantages like its portability, not only with native operating systems but also mobile (Android), and innate memory management and exceptions handling **(3)**.

Since the scope of this project is the implementation of part of the DomoBus technology, no external libraries are used and standard ones limited in order to reduce the complexity and requirements of the developed library. Additionally, no graphical interfaces are added, with the few external interactions made directly through the console running the application.

2.2.2. Operating System

Since it uses a multi-platform programming language, the range of supported operating systems are wide. But project was implemented and tested using the Microsoft Windows 7 OS, 64-bit version. However, in the very few cases that could generate conflict when used in other platforms, automatic verifications and modifications are appropriately made. Such as paths corrections and commands corrections.

Additionally, during the test phase, the multiple Supervisor applications are ran and communicate locally.

3. Description of the Solution

Basically the idea of the presented solution is to build a communications module which can be internally associated together with a supervision module, within any Supervisor application.

The interactions across all levels of the DomoBus network and between any application and device are made possible without any confusion by identifying each participating node with a unique and global address. More specifically, a 2-bytes integer for applications of the Supervision level. And for devices, a 4-bytes combination of its Supervisor gateway 2-bytes address and a 2-bytes Control-Level address **(1)**.

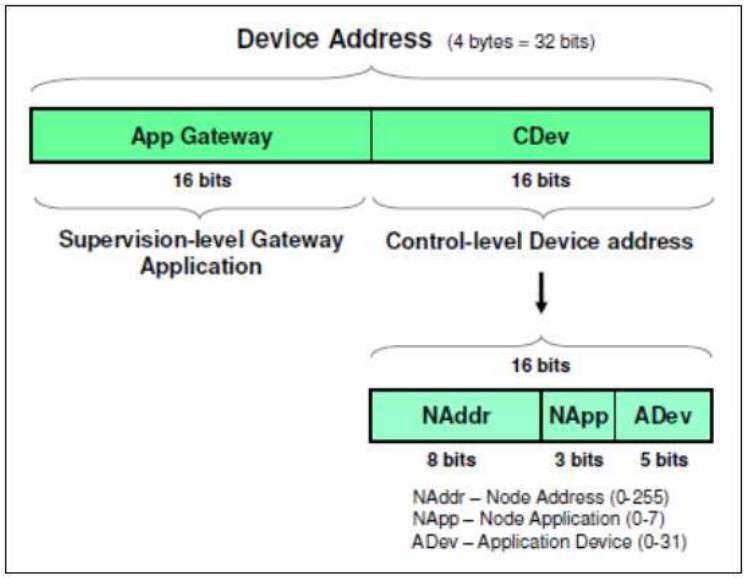


*Figure 1 - Device address structure **(1)***

3.1. Initial Approach

The initial approach was to regroup the functionalities needed for the communications module into three main, intercommunicating, components, as shown in the figure below:

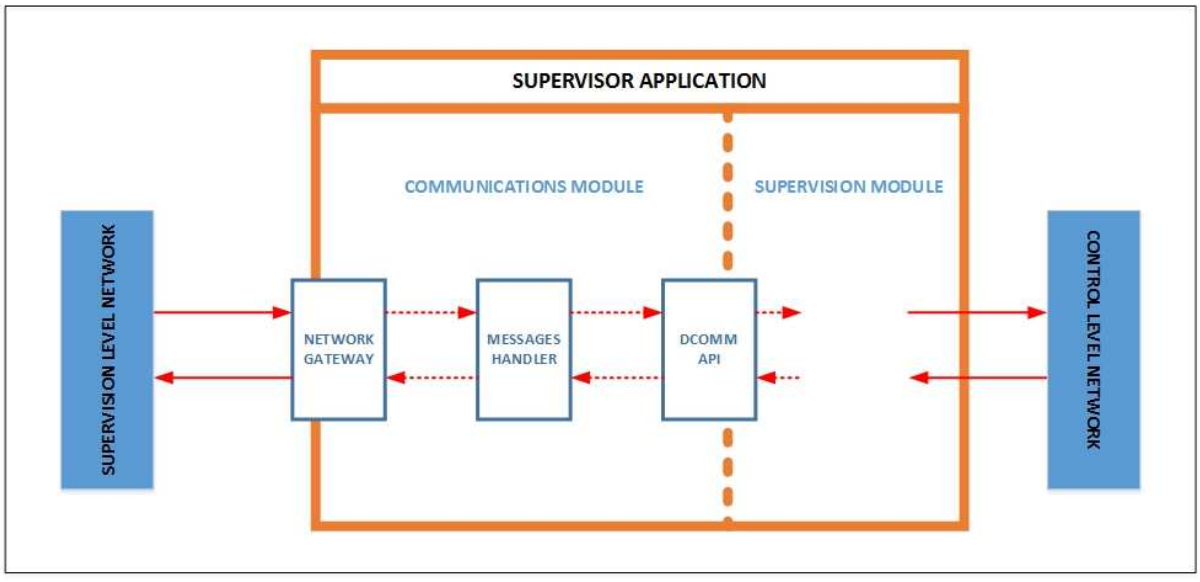


Figure 2 - Global View

This diagram is not intended to reflect the entirety of the internal structure of an actual Supervisor application, as other components like the Control Level gateways, user interface and the supervision module itself is outside the scope of this project; and rather a limited view evidencing the existing interactions that surrounds the communications part of a Supervisor.

Components:

- Network Gateway: would be responsible for managing all network connections and data exchanges with the other peers present at the Supervision Level, effectively connecting not only the Supervisor, as well as all the devices forming the locals DomoBus Control Network connected to it.
- Messages Handler: would process and analyze all commands and other messages being exchanged between the devices and applications through both network levels. Converting the data into the different formats when appropriate.
- DCOMM API: the DomoBus communications interface which serves as a bridge between both modules, and more generally between both network levels. Providing synchronous and asynchronous methods and callbacks for executing and transmitting commands mostly to end-devices and sometimes other Supervisor applications.

In the next chapter, we can see that the actual organization of the classes composing the module, don't portray a complete and strict separation as shown with each component, with some overlapping across multiple roles.

3.2. Components

3.2.1. Network Gateway

The Network Gateway component has multiple functions, applied with the core role of linking the application with the outside world, which in reality is either the Internet or a local private TCP/IP network.

It manages connections with all other known Supervisors in order to quickly identify incoming packets and transmit outgoing packets to the given addresses. Furthermore, it continuously listen on its shared network port for messages from new connections, controlling their provenance and specified identity. As to enable dynamically changing DomoBus networks, the gateway registers the Supervisor address with a global DomoBus Name Server, referred to as DNS, and supports sending requests and handling responses. This allows new Supervisors, and by extension new Control-Level networks, to join the DomoBus network. Also this way, each time a Supervisor receives a message from an unknown application address, it can simply query the DNS to confirm the identity of this unknown party, effectively adding a basic level of security.

Other of its functions center around providing the missing services excluded from the UDP transport protocol used by the Supervision Level network for exchanging data, which can originate certain transmission problems, namely:

- Reliable transmission;
- Error-checking;
- Flow control.

These services are already thought of and given guidelines in the existing DomoBus specification documents **(4)**.

Specifically, reliable transmission is reached by using a packet retransmission protocol: each outgoing packet is given a finite number of times it can be retransmitted. Retransmissions stop when an acknowledgement, response packet, has been received or it has reached the maximum retries.
Flow control goes in par with reliable transmission as it allows to determine the corresponding acknowledgements and detect and discard duplicates packets received. In practice this is accomplished by using sequences numbers, present in each exchanged packet. Each pair of Supervisors uses its own sequence, as to reduce the possible ordering conflicts that can happen in a busy network.
Finally error-checking serves to ensure the integrity of the data exchange and is achieved firstly with the first byte of a Supervision Level packet reserved for indicating the total data length and more importantly, the last byte which contains the result of a shared checksum function applied to its contents.

New incoming packets that pass those basic verification mechanisms are then internally put into pending queues for the Messages Handler component to process.

3.2.2. Messages Handler

The main role of this component is the conversion of the data being exchanged between the other two components and in both directions.
It is in here that the incoming packets are thoroughly analyzed, with new commands separated from responses and operation types verified, in order to correctly send them to the supervision module through the API using the corresponding callback or process locally what needs to be.

Outgoing commands, be it new request or responses, who have their contents emanating individually from the API, have first to be rearranged into Supervision Level packets. The Handler also constructs the appropriate control field, toggling the required bits, and sets the correct destination and origin addresses.

Furthermore, the registration of logs is mostly done within this component as it can give a centralized overview of the events occurring between both network levels that it connects.

3.2.3. DCOMM API

DomoBus has originally 4 different types of commands defined, **GET**, **SET**, **NOTIFY** and **EXEC**; but has support for up to 8 **(1)**.
However in this project we introduce a fifth one, **DCOMM**, intended to be exclusively used at the Supervision Level, mostly between Supervisors but also for their interactions with the DNS. It uses the same format for its contents as the **EXEC** commands. This fifth command enables then a separation of the messages during processing, especially from the **EXEC** types, and thus allows a greater range of provided services as each keep 256 possible different functions (as allowed with a 1-byte field).

Command	OpCode CTR bits	Description
GET	000	requests the current value of a device's property
SET	001	updates the value of a device's property with the given value
NOTIFY	010	notifies the current value of a device's property to a subscribed application
EXEC	011	requests the remote execution of a given function
RESERVED	100-110	reserved operations for future use or custom services
DCOMM	111	Supervision Level commands

Table 1 - DomoBus supported operations

Therefore and as expected, the DomoBus communications interface has to provide methods for, sending any of these supported commands, when looking at the perspective of the supervision module and the Control Level network of the Supervisor, and receiving them for local processing, from the Supervision Level network.
In par with the motivation of having a generic, customizable framework, it should also provide additional methods for dealing with remaining unspecified operations, identified with the suffix **RESERVED**. This allows the redirection of such commands, in the instances when it is wanted, and such permits given Supervisors to offer special services if need to be without further modifications or incompatibilities.

And while all the requirements for expanding the services Supervisors wish to provide are present within the API, a key element is that each Supervisor only has to implement the ones they actually want offer and support. Thus lowering its limitations and restrictions.

Additionally, it supplies methods for all outgoing commands in both a synchronous and asynchronous manner. The synchronous methods intended to block and wait until the response to the command arrives, or a timeout occurs. While the asynchronous methods simply transmit the command and return, with an eventual response being handled posteriorly. The following table displays the identifiers of these methods referred above. To distinguish them, the synchronous methods contain the token "sync" attached to their names.

Asynchronous Methods	Synchronous Methods
DComm_send_msg_GET	DComm_send_sync_msg_GET
DComm_send_msg_SET	DComm_send_sync_msg_SET
DComm_send_msg_NOTIFY	DComm_send_sync_msg_NOTIFY
DComm_send_msg_EXEC	DComm_send_sync_msg_EXEC
DComm_send_msg_RESERVED	DComm_send_sync_msg_RESERVED
DComm_send_msg_DCOMM	DComm_send_sync_msg_DCOMM

Table 2 - API outgoing command methods

On the other hand, the methods for handling incoming commands are identified as callbacks and differ from some of their counterparts. For starters, there is no callback for the **DCOMM** commands, as they are processed at the communications module side. Also the callbacks distinguish GET commands between new requests and actual responses to past requests, **DComm_process_callback_ANSWER_GET**. And finally an extra method is provided for handling commands errors, which either occurred during transmission or were sent through a response, **DComm_process_callback_ERROR.**

Callback Methods
DComm_process_callback_GET
DComm_process_callback_ANSWER_GET
DComm_process_callback_SET
DComm_process_callback_NOTIFY
DComm_process_callback_EXEC
DComm_process_callback_RESERVED
DComm_process_callback_ERROR

Table 3 - API incoming commands methods

4. Details of the Solution

4.1. Implementation

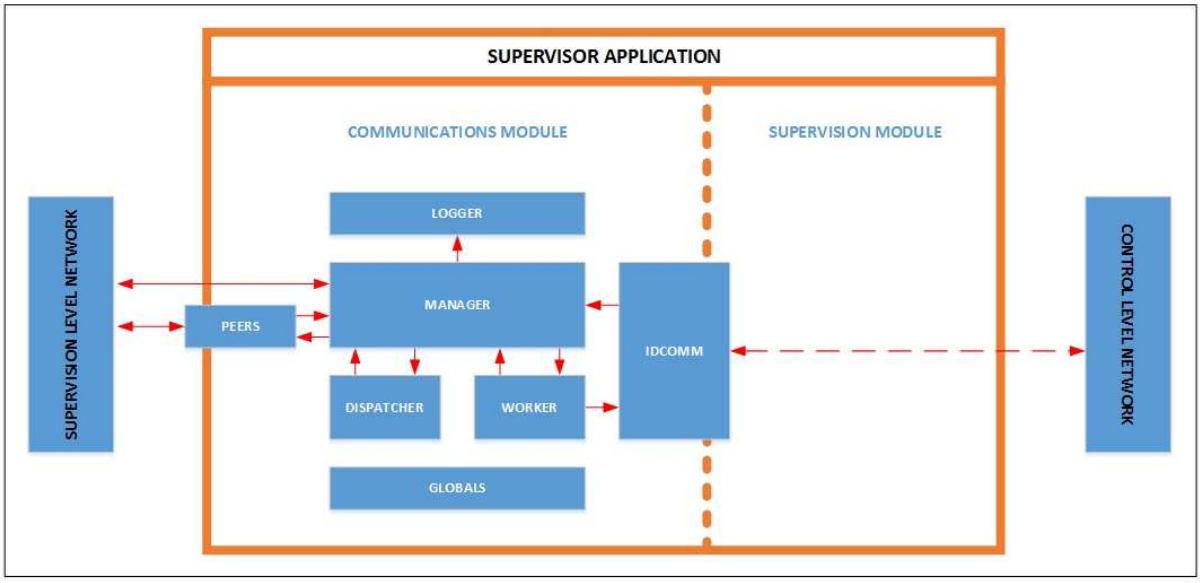


Figure 3 - Class structure

4.2. Library Classes

4.2.1. IDComm

This is the abstracted DomoBus communications interface which defines all the methods of the proposed solution stated in the previous chapter of this report.
For each method it specifies the necessary arguments and their data types and the expected outputted data type, if applicable.

For all outgoing commands, those being sent, on behalf of the supervision module, to other Supervisors, contain an additional input argument, a Boolean indicating the priority requested for the given command. This relates to the application of the priority bit contained in the control field of all Supervision Level packets, as defined in **(1)**. All the specifications of these methods are present in the Annexes of this report.

Additionally it contains a custom Exception, called **UnsupportedCommandException**, which is exactly used for allowing Supervisor the free will of choosing which services they can provide and support. In practice with an implementation of the API, unwanted methods just have to throw a new instance of this exception as to inform, has the name portrays, that it isn't currently supported.

The fallback and handling of this exception is already built within the library. Such events when they occur are registered and a response to the command received is dispatched. The response goes with the acknowledge and error bits of the control field toggled and contains an error code specific to this problem, **ERROR_UNSUPPORTED_COMMAND** as identified in the Globals class presented below.

Both modules of a Supervisor application have to implement its methods. All the callback methods for the supervision module, while communications module, through the Manager which is detailed below, stays with the send methods.

4.2.2. Manager

This is the principal class of the library and, as the name implies, functions as a literal manager of the communications module. It actually instantiates all the other classes used and operates across all components proposed earlier.

Its functions are to manage the subscription system and peer connections, listen to its public port, interact with the DNS, implements the IDComm methods and serves as an intermediary between some of the other classes.

The Manager implements all outgoing command methods of the DComm interface.
For asynchronous commands, it simply sets the corresponding operation code on the control field, appropriately concatenates the command contents and then finally transfers all the data to the Dispatcher for transmission.
With synchronous messages, their execution is longer and blocking. Those methods extend what was implemented for the asynchronous ones, however it only asks the Dispatcher to create a Supervision Level packet instead of letting it also handle the transmission. Afterwards, it verifies if the target application is known, sending a DNS request if necessary. Finally, it proceeds to fetch the Peer object linked to that application and order it to send a synchronous message. The Manager only blocks until the Peer object either returns with a response, which can be an error, or a timeout is issued. It then forwards the response or, in the latter case, creates and return an error response composed of a byte with the error bit toggled and a byte array with only the error code corresponding to ERROR_TRANSMISSION_FAILED.

For both NOTIFY requests, the proceedings differ slightly from the above. First the Manager verifies if there actually is at least one subscriber of the indicated device's property. If not, it stops and, for synchronous requests, returns an acknowledge byte confirming the successful execution of the request.
It then completes the property description field according to the current value, as defined in **(1)**. And proceeds to send the NOTIFY command to all subscribed applications, following the corresponding manner stated before. An additional change during synchronous calls is that a positive response is only given when all subscribers have acknowledged the notification.

DCOMM commands also have an additional step, if they are destined to itself, they are directly sent to the Worker for processing and its response transferred back.
When processed locally, two errors can happen, either DCOMM commands are no implemented, ERROR_UNSUPPORTED_COMMAND, or there was an actual error during its execution, ERROR_DCOMM_COMMAND.

The incoming command methods of the interface are exclusively implemented by the supervision module, who provides a reference of that implementation, the API, to the Manager. However the Manager also provides a bridge for calling the incoming method for handling errors, DComm_callback_process_ERROR, only to catch an eventual UnsupportedCommandException if it isn't implemented by the supervision module. A reference to that API is also given to the Worker, facilitating the calls it require.

The Manager also creates additional bridges, methods used to call other objects' methods, for the Logger, Worker, Dispatcher and Peer objects. The bridges are mostly used to reduce the complexity of the classes and avoid redundant references, also updates only need to be made in the Manager instead of all other classes using a given bridge method.
For the Logger, it permits calling the Log method, for registering events. For the Worker, it has a bridge to forward new messages for processing, AddMessage. In the case of the Dispatcher, it enables two calls. One for transmitting an incoming packet, NewMessage, and the other one, to create and send a new packet, CreateMessage. A Peer object is created individually during instantiation for each known Supervisor present in the configuration file, and it can be updated afterwards as needed. For these objects, two bridge methods are provided to send messages, asynchronously or synchronously.

The Manager runs a separate thread for listening to its public port. Within this thread, it can receive new packets from the Supervision Level network, either originating from an unknown or recently joined Supervisor, in which case a DNS query is sent through its DNSRequest method and awaits for a

retransmission; or from a known peer, in this case the socket of the corresponding Peer object is updated and the message sent to the Dispatcher for further verification. As the Manager only checks the packet's length, the specified application address of origin and the remote network's address retrieved from the socket.

Another functionality running in a separate thread is the update task for the configuration file. Every 30 minutes, if there was any changes since the previous update, new properties are added and old ones removed from the local configuration file to reflect the current state of the Supervisor's relationships with the rest of the network. Currently only adding new Supervisors or updates pertaining to the subscriptions can actually happen.
Also, the configuration file is only loaded once, during the instantiation of the Manager, to retrieve its DomoBus address, its public port and, eventually, mappings of other Supervisors, subscriptions and the contact of the DNS. The configuration file is detailed further in this chapter.

As to efficiently manage communications at the Supervision Level, the Manager keeps the sequences numbers used and provides all the required methods to manage them from within the Worker or the Dispatcher. Each pair of Supervisors follows its own sequence and therefore, two maps are used to store the relevant ones. One map keeps the latest sequence number used and the other all the sequence numbers pertaining to commands, sent or received, currently waiting to be processed, as to discard duplicates.

As mentioned before, interactions with the DNS are made here. Other than sending requests for unknown peers, during its instantiation, the Manager also registers its Supervisor with the DNS, updating eventual changes of IP address or port used. The formats of the packets exchanged during these interactions were not specified and such the details of the formats used are present in the Annexes of this report, however they are a subset of DCOMM commands, which in turn are based on the EXEC commands.

Subscriptions are also managed dynamically, they are loaded during instantiation and kept into a map of publishers representing devices from the Control-Level networks. In turn each of device contains another map with the subscribed properties and their subscribers list. The Manager provides three methods to update the subscriptions. Two to add or remove a subscription and a third one to remove all subscriptions of a given application. These methods can be called from the worker after processing DCOMM commands related to the subscription system. Those DCOMM commands were defined specifically for this project and are also detailed in the Annexes of the report. DCOMM responses, unless stated otherwise, follow the same format as the other commands.

4.2.3. Worker

The Worker is instantiated by the Manager and receives a reference to it and the supervision module's API. Apart from the error callback mentioned earlier, all API callbacks are transmitted through the Worker, as its role is to process incoming messages, new requests or responses containing values.

The Worker stores the incoming messages, received from the Dispatcher through the Manager, into two queues according to their priority bit. Since the queues have an upper limit, as intended in **(1/4)**, priority packets have a preference over both queues, first the priority queue is checked, in second the normal queue and if both are full, only then it is discarded. While normal packets can only be put into the normal queue, up to its limit, before being discarded. The Worker's method for keeping now packets returns false when they are discarded, allowing the Dispatcher to send back a reply indicating the error, **ERROR_QUEUES_FULL**.

A specific thread loops over the queues to process the pending packets, going through the priority ones first. Each packet is separated according to its operation code and processed adequately. Answer packets are also separated, especially useful for **GET** commands as those responses use a different method from the API, and their contents are extracted into individual variables as required.

Afterwards the commands are logged and API callbacks executed. The Worker also engross the responsibility of preparing the responses, when applicable, or possible errors that occur and transmitting them to the Dispatcher.

However DCOMM commands, except the listing of all publishers, are locally executed by the Worker. Commands for listing the available publishers of the managed Control-Level networks are transferred to the supervision module as an EXEC command. Currently only DCOMM commands relating to the DNS interactions and the subscription system are used, by only the subscriptions ones actually reach the Worker and are implemented.

Another functionality of the Worker regards the processing of new SET commands. After being processed and transmitted through the API, if the response doesn't contain an error, the notification method of the Manager is called to transmit NOTIFY commands with the new value to all subscribed applications.

4.2.4. Dispatcher

The Dispatcher's main role is to manage communications, especially the transmissions.

It maintains structures with the messages being sent and retransmission queues.
The queues help to keep track of the number of possible retransmissions left and maintain an order to follow. It uses a specific thread to process those queues and dispatch the packets, through the Manager, using the corresponding socket. When the target Supervisor is not found, it also asks the Manager to complete a DNS request asynchronously.
To avoid instantaneous retransmissions, a small delay is introduce and each packet actually has a retransmission time set.

Another important feature is the creation of Supervision Level packets, it concatenates the arguments and correctly sets some fields, like the total length and the checksum.

Incoming packets, caught either by the Manager or a Peer object, and forwarded here. The Dispatcher then verifies a number of things: the packet's length, its checksum, the destination's address and sequence number. If it detects any error, the packet is discarded.
It checks for responses, removing the corresponding requests from the retransmission list and transfers them to the Worker. Generic command errors are handled and the error callback called via the Manager, while the rest are also forwarded to the Worker.
Non-response packets that have a distinct and valid sequence number are, just like responses, given to the Worker for further processing.

Generic command errors englobe ERROR_UNSUPPORTED_COMMAND and ERROR_[type]_COMMAND, where the type can be any of the existing operations or comport the RESERVED keyword.

4.2.5. Peer

A Peer object specifically handles a connection with a given Supervisor. When initialized, it receives a reference to the Manager, the target application address, its IP or hostname and the public port. A UDP socket can also be specified, otherwise one will be created before running.

When running, it continuously listens to its bound Supervisor in an exclusive thread, reading new packets and forwarding them to the Dispatcher using the Manager's reference.

Each Peer object has two methods, which were addressed earlier, for synchronous and asynchronous transmissions.
The synchronous method is blocking and locks the socket for itself until the timeout occurs or the response arrives. An additional measure was implemented to avoid returning an incorrect response, since during the lapse of the synchronous transmission the target Supervisor could have sent packet not corresponding to this request. As such the Peer object waits until receiving a response pertaining

to the request's sequence number, while forwarding the other received packets the same way they are within the thread. The returned data is built as described in the Annexes, however If the timeout happens, the Peer object returns a byte with the error bit toggled and a byte array composed of the timeout error, **ERROR_TIMEOUT**, inside an object array of 2 elements.

4.2.6. Logger

The Logger is a simple class which sole role is to keep an up-to-date opened log file and register now entries into it. In this sense, it schedules a periodic task to check the date of the current log file, as to create a separate file for each day.

All log files are stored in a specific folder, currently called "**Logs**", whose existence is ensured during its instantiation. It also initially either open and appends to the corresponding log file or creates a new one if inexistent.

The Logger supplies a method called Log for registering events, that the Manager provides a bridge for, which only requires its category and details. The Logger then affixes a timestamp and formats correctly the new entry.

4.2.7. Globals

As the name implies, this is a purely static class used by all the other classes. It contains all used constants, acting as global variables, and even a couple of auxiliary functions.

Some of the variables where already defined in **(1)**, while others, completely new or simple with different values, emanate from initiatives took during the implementation of this project. Their scope range from error codes, operation codes, control field options, configuration properties, queue sizes, waiting periods, et al. The complete list is also detailed in the Annexes.

The auxiliary functions provided are associated to 3 groups. A single one allows to check if a given bit is set.
Two functions different implementations for the calculation of the packet's checksum, the 8-bits version of the cyclic redundancy check (CRC) and a simple checksum with an offset.
The remaining two are conversion functions between integers to and from byte arrays. The endianness is specified in a variable, allowing the library to rapidly adapt as necessary. They are quite versatile, allowing to specify positions offset, however currently they only support lengths of 2 bytes. This is an intended limitation since Supervision Level packets only encode at most integers of that size. Some values might be exceptions to this, but they are transferred to the supervision module and therefore fall out of the scope of those functions.

4.3. External Data

4.3.1. Configuration File

An external configuration file is used for loading certain information, pertaining to an individual Supervisor and its relations with the DomoBus, onto the application. Its default name is "**dcomm.cfg**", however a different name can be supplied in the conditions to be presented afterwards in this report. It is composed of properties, which formats follow the guidelines indicated in the DomoBus documents. However, over the course of this implementation, an additional property has been added for specifying the DomoBus Name Server contact within the Supervision Level network, called **DNS**.

Both **APP_ADDR** and **APP_PORT** properties are mandatory, as their omission causes an impossibility for the Supervisor to properly operate within the network since the other Supervisors wouldn't be able to communicate and therefore respond to commands. In case of repeated presence, the last entry of each property is kept.

Property	Description	Use
APP_ADDR	Specifies the application address used by the Supervisor in the DomoBus network it belongs to.	APP_ADDR [appAddr]
APP_PORT	Specifies the public port the Supervisor will be listening to for communications at the network's Supervision Level.	APP_PORT [port]
ADDR_MAP	Indicates the identification of a known Supervisor.	ADDR_MAP [appAddr] [IP/hostname] [port]
SUBSCRIBE	Specifies a subscription from a peer Supervisor for the value of the given device's property.	SUBSCRIBE [devAddr] [propertyId] [appAddr]
DNS	Designates the network contact of a DomoBus Name Server.	DNS [IP/hostname] [port]

Table 4 - Configuration properties

In the current implementation, the network addresses can be specified in either IP versions, IPv4 and IPv6, or ultimately, indirectly using a hostname.

Public ports are not fixed to a certain number, allowing multiple Supervisors to operate concurrently within the same machine. The issue of uncertainty over the port number used is solved with the use of the DNS, it even renders possible changes over time that suit the needs the applications with the registration command during their start up.

All changes that occur during the operations of a Supervisor and external interactions about the peers mapping and subscriptions, are reflected and persisted onto the configuration file through the Manager's Update Task.

4.3.2. Log Files

They are created daily and managed by the Logger class, as defined in the previous point. Their utility is to provide auditing and monitoring capacities over the services provided by the application and interactions that go through it between both network's levels.

Each entry of a log file represents an event that occurred. This event is composed of the time of occurrence, the category of events it belongs to and its details. In the version of library, 4 base categories were defined. All entries are separated by a newline and their contents by a single tabulation, making the files viewing possible with any spreadsheet-oriented software (Microsoft Excel, LibreOffice Calc, et al).

Category	Description
ERROR	Regroups all occurrences of errors that happen either internally or resulting from the processing of commands.
COMMANDS	All commands, received or sent, that go through the Supervisor or are locally processed, when appropriate.
DNS	All events related to the interactions between the application and the DNS.
SYSTEM	Events pertaining to the internal services, their statuses, initializations and shutdowns, throughout the Supervisor's operating period.

They are named after the current date, "YYYY_MM_DD", and use the file extension ".log".

4.4. Generic Library

The generic library is the intended release version of this project. It is self-contained and can be integrated to any other project or, considering the objective of this implementation, the supervision module of a Supervisor. The only missing element is the configuration file, which cannot be included in a library JAR, and such needs to be created by the Supervisor administrator. However it was completely detailed in the previous subchapter and an example will be provided in the integration guide.

4.4.1. Creating the Library

Creating the JAR file of the DomoBus communications library is a simple procedure. First we have to create our own "MANIFEST.MF" file, which provides basic information about the library, like the java version used, the author and package name. The following file was used:

```
Manifest-Version: 1.0
Created-By: 1.7.0_15 (Oracle Corporation)
Name: domobus/communications
Built-By: Rafael Afonso Rodrigues
```

Afterwards we proceed to create a folder named "domobus" and another one called "communications", to which we copy our 7 developed classes. We end up having the following tree structure:

```
domobus
|--communications
   |--Dispatcher.java
   |--Globals.java
   |--IDComm.java
   |--Logger.java
   |--Manager.java
   |--Peer.java
   |--Worker.java
MANIFEST.MF
```

Then, we compile all of those files, creating **.class** files from our source code, using the **javac** tool within the communications folder and specifying a wildcard character followed by the **.java** extension:

```
javac *.java
```

Upon completion, we can remove the initial source files, those with the **.java** extension, as to leave only the compiled classes inside our folder. We end up with more files than our 7 original files because all additional objects created, like enumeration variables, customized exceptions and threaded tasks, are separated from the main class they were implemented with.

```
domobus
|--communications
   |--Dispatcher$1.class
```

```
            |--Dispatcher.class
            |--Globals$1.class
            |--Globals.class
            |--IDComm.class
            |--Logger$1.class
            |--Logger.class
            |--Manager$1.class
            |--Manager$2.class
            |--Manager.class
            |--Peer.class
            |--UnsupportedCommandException.class
            |--Worker.class
        MANIFEST.MF
```

Finally through the command line, we create the library by executing, inside the folder containing all of the above, the following command:

```
jar cfm DCommLibrary.jar MANIFEST.MF domobus
```

Where **jar** is an executable tool provided by the Java SE Development Kit, **DCommLibrary.jar** the desired name of our library and **cfm** the options indicating we want to create a new JAR file and provide our own manifest file.

The library is finally created inside the same folder and ready for deployment.

4.4.1. Integrating the Library

Integrating the library is quite straightforward, an administrator simply has to add the **DCommLibrary.jar** to its project. Most IDE have a direct option to add JAR files, as exemplified in the figure above through NetBeans IDE.

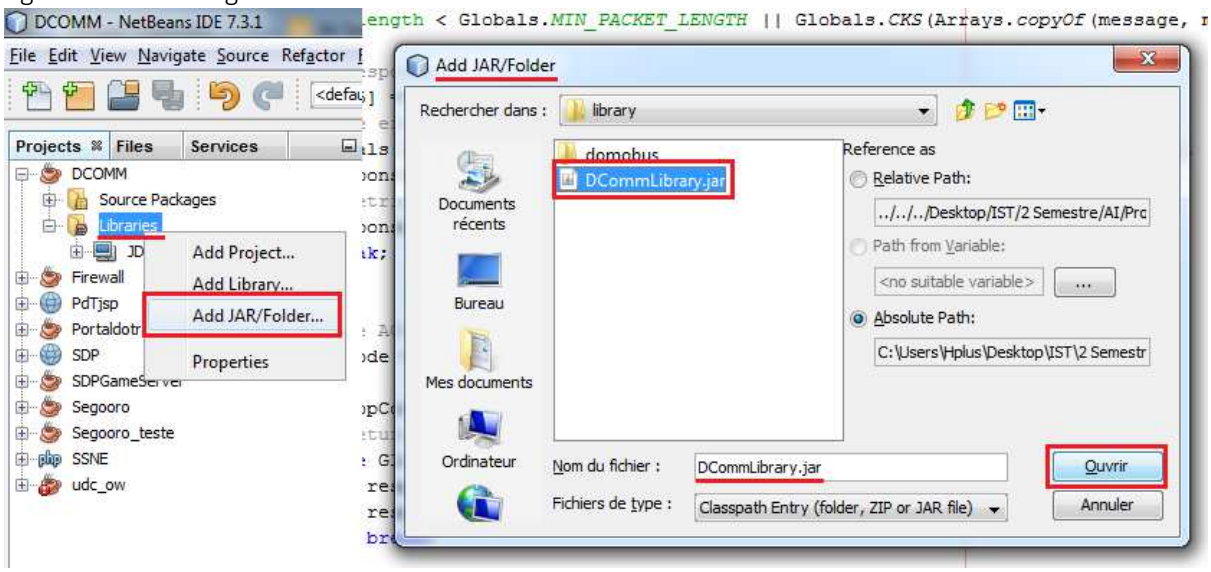


Figure 4 - NetBeans: adding DComm library JAR

After adding the library, we can navigate through its contents and verify it really is the DComm library.

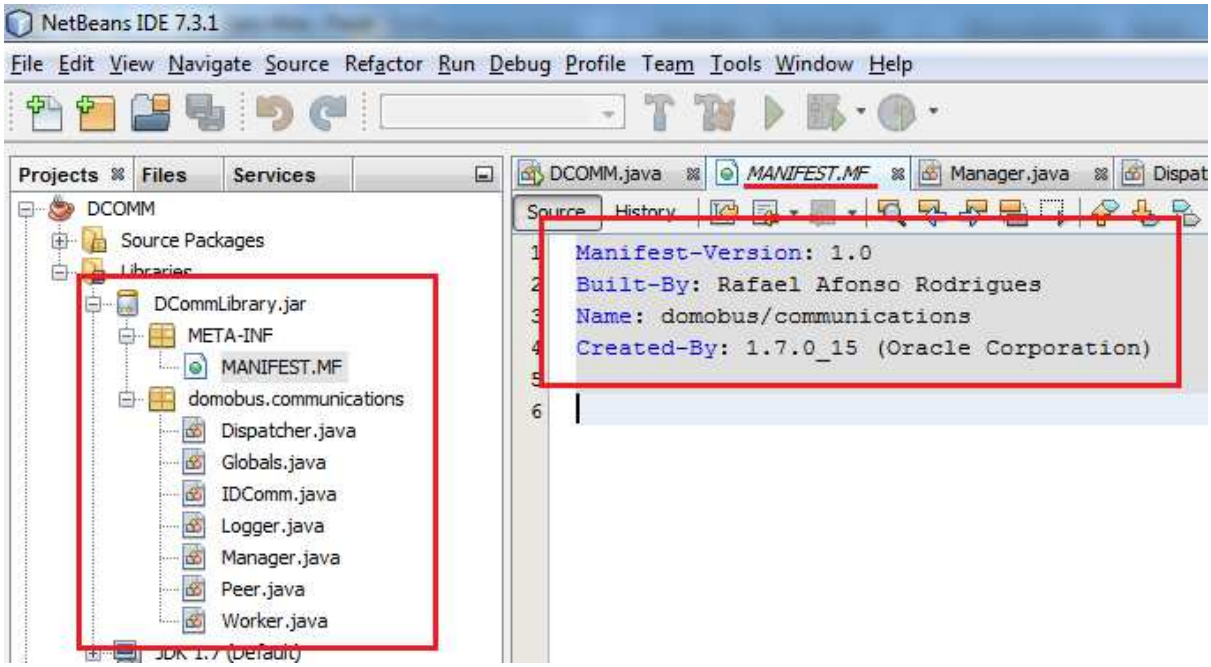


Figure 5 - NetBeans: DComm library contents

As mentioned before, the Manager class is at the center of the DomoBus communications library, thus it was implemented in such a way that in order to integrate the communication module to another of the Supervisor's modules, we actually only need to instantiate that class. But first to be able to reach it, we have to import it from within the project integrating the library. This is accomplished by making the following import:

```
import domobus.communications.Manager;
```

Any of the other classes, like Globals which can be useful even for other modules, can be reached and utilized by using that import prefix, which basically corresponds to the library's package name.

Afterwards we can instantiate the communications module Manager. Looking at the Manager class, we can observe that it only requires two arguments, the supervision module's implementation of the DComm interface and the configuration file, the default name can be retrieved from the Globals class.

```
public Manager(IDComm API, final String configurationFile) throws FileNotFoundException, IOException, InstantiationException
```

Figure 6 - Manager class: instantiation

Thus the following code is enough accomplish this:

```
Manager manager = new Manager(API, configurationFile);
```

However, as stated before and shown above, the Manager class can throw multiple exceptions if an argument is invalid. Therefore a more complete example is presented below.

```
IDCommImpl API = new IDCommImpl(); //Implementation class of the DComm interface
String configurationFile = Globals.DEFAULT_CONFIG_FILE;

try {
    Manager manager = new Manager(API, configurationFile);
}
catch (FileNotFoundException ex) {
    System.out.println("Configuration file not found.");
}
catch (IOException ex) {
    System.out.println("Exception while reading from configuration file.");
}
catch (InstantiationException ex) {
    System.out.println("Currupt configuration file, missing mandatory properties.");
}
```

Figure 7 - Manager class: instantiation example

4.4.2. Starting and Stopping Services

With a created Manager instance, an administrator can start and stop all the services provided using only 2 lines of code, as it has two methods rightly for that effect.

Launching the services is accomplished by calling the **start** method of the Manager (since it extends the standard Thread class, **start** calls the implemented **run** method). This will not only start its own services but also all of the other threads, by calling their own **start** methods, and register itself with the DNS.

```
194         @Override
            public void run() {
196             try {
197                 if(DNSRegistration()) System.out.println("Registered with DNS.");
198
199                 //start Worker
200                 worker.start();
201
202                 //start Dispatcher
203                 dispatcher.start();
204
205                 //start listening to known peers
206                 for(Peer peer: peerList.values()) peer.start();
207
208                 //start accepting new connections
209                 DatagramSocket serverSocket = new DatagramSocket(port);
210                 byte[] receiveData = new byte[Globals.MAX_PACKET_LENGTH];
211                 while(running) {
212                     //wait for a new packet
213                     DatagramPacket packet = new DatagramPacket(receiveData, Globals.MAX_PACKET_LENGTH);
214                     serverSocket.receive(packet);
```

Figure 8 - Manager class: start services

To shut down all operations, the application simply has to call the Disconnect method, this will in turn call the nested Disconnect methods, implemented for the other classes, and stop the active threads, close opened connections and set to false any helping variable.

```
174     public void Disconnect() {
175         //Stop dispatching messages
176         dispatcher.Stop();
177         System.out.println("Dispatcher stopped.");
178         //Stop accepting new connections
179         running = false;
180         //Close open connections
181         for(Peer peer: peerList.values()) peer.Disconnect();
182         System.out.println("Stopped incomming connections.");
183         //Stop processing messages
184         worker.Stop();
185         System.out.println("Worker stopped.");
186         //Close logs
187         logger.Stop();
188     };
```

Figure 9 - Manager class: stop services

4.5. Test Version

The test version is an implementation of the DCommLibrary.jar with the objective of testing the developed communications module. In this version we were required to implement the API methods for the supervision module side and some user interaction capabilities has to allow controlling it.

Additionally more information is displayed directly to the user through the console. However it is purposively limited, encompassing only the highlights of what is currently happening, because it could confuse the user while he's interacting with the program.

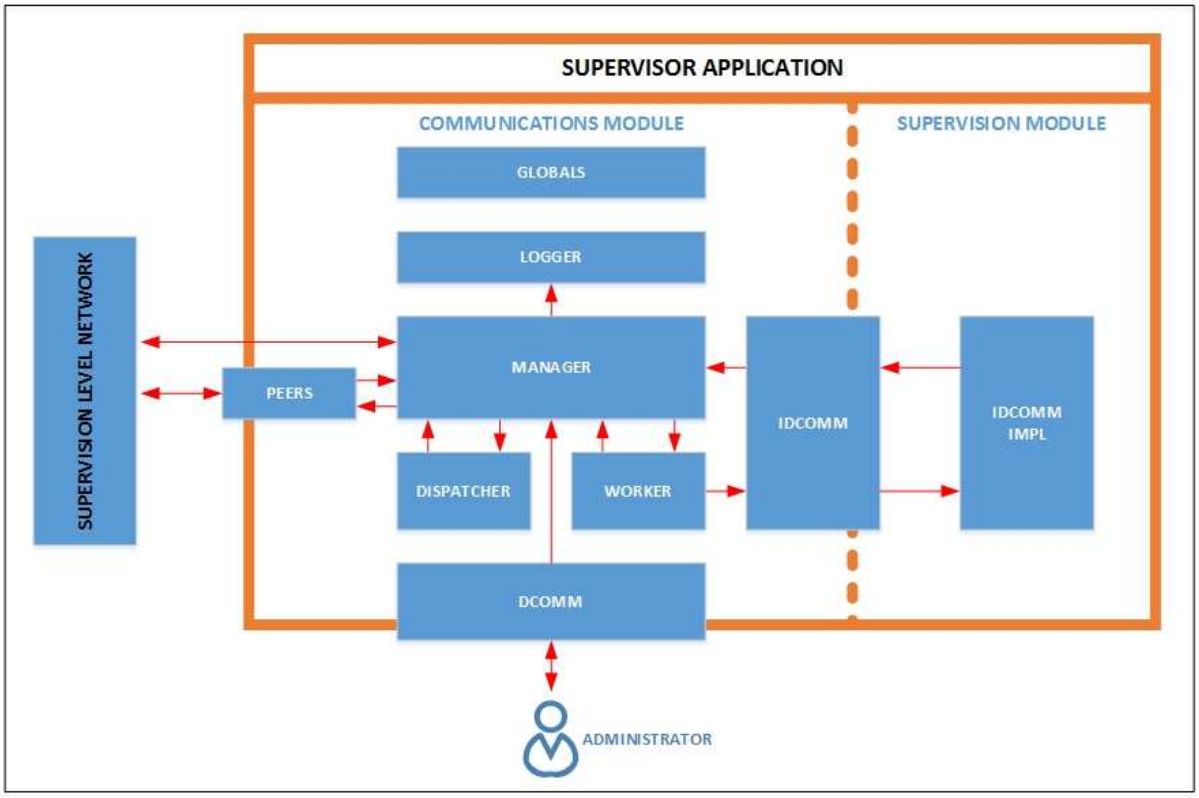


Figure 10 - Test Version: class structure

4.5.1. Additional Classes

As observable in the Figure 10, the test version contains two additional classes, **IDCommImpl**, which is the API or implementation of the DComm interface of the simulated supervision module; and **DCOMM**, which is the main class of the project and the one being executed.

This API implements all callback methods except for the **DComm_callback_process_RESERVED**, in which case it throws the **UnsupportedCommandException**. All implementations are quite short as they are only intended to log and provide some small feedback by also displaying the arguments of the commands it receives into the console. For the non-void methods, it returns a byte array with 1 random integer element when **DComm_callback_process_GET** is called; an acknowledge byte for **DComm_callback_process_SET**; and a true Boolean after a **DComm_callback_process_NOTIFY** call.

The **DCOMM** main class is quite more complex as it offers some additional services, supports runtime arguments, handles possible exceptions and the instantiations and is able to provide simply interactions through the console with a user.

4.5.2. Additional Services

The first small functionality it provides is supporting runtime arguments passed when it is execute. For now, only three options are available: specify a configuration file, specify a commands file to execute and print help mnemonic.

Option	Description	Runtime Arguments
Help	Prints the program's help.	-help
Configuration File	Specifies a different configuration file.	-config [configuration file]
Commands File	Specifies a commands file to be execute right after starting the manager.	-execute [commands file]

These options are demonstrated in the next point of this subchapter and are purely optional. When used however, the files existence are verified, indicating an error message in case of failure.

An important service provided is the ability of executing a list of commands directly from a file. Files can either be specified at runtime or successively through the console. DCOMM can process all five available commands from a file, **RESERVED** types are excluded, supports comments and provides a specific command for setting the wasted destination address. However each command has to be written in a newline and when an unrecognized type is read from the file, it simply indicates that it is not supported. The first command must specify a destination address, but it can be changed afterwards. The table below shows the all the supported commands and their formats.

Command	Description	Use
SD	Specifies the target's destination address	SD [appAddr]
GET	Sends a GET command	GET [devAddrOrig] [propDescOrig] [devAddrDest] [propDescDest]
SET	Sends a SET command	SET [devAddrOrig] [propDescOrig] [devAddrDest] [propDescDest] [value]
NOTIFY	Sends a NOTIFY command	NOTIFY [devOrig] [propOrig] [value]
EXEC	Sends a EXEC command	EXEC [command] [arguments]
DCOMM	Sends a DCOMM command	DCOMM [command] [arguments]
# or ; or //	comments	#... OR ;... OR //...

Table 5 - Commands File format

A user can also manually and individually write any of those five operation types. In order for a user to interact with the program and benefit from those functionalities, the test version supports three basic commands:

- **CMD**: to execute a single command, specifying its contents afterwards;
- **EXECUTE**: execute the file indicated;
- **EXIT**: shuts down all services by calling the **Disconnect** method of the Manager.

If he chooses the single command, the program will ask for an application address, if it's the first time or the user wants to alter it, and then gives the choice between all command types.

All commands a user can request through the DCOMM are execute asynchronously, thus it will not block and only display the response, or even incoming commands from other applications, posteriorly.

4.5.3. How to Use

The test version can be used by launching the compiled DCOMM.jar like any other executable JAR file.

```
java -jar DCOMM.jar
```

Using the runtime help flag would lead to the following statements:

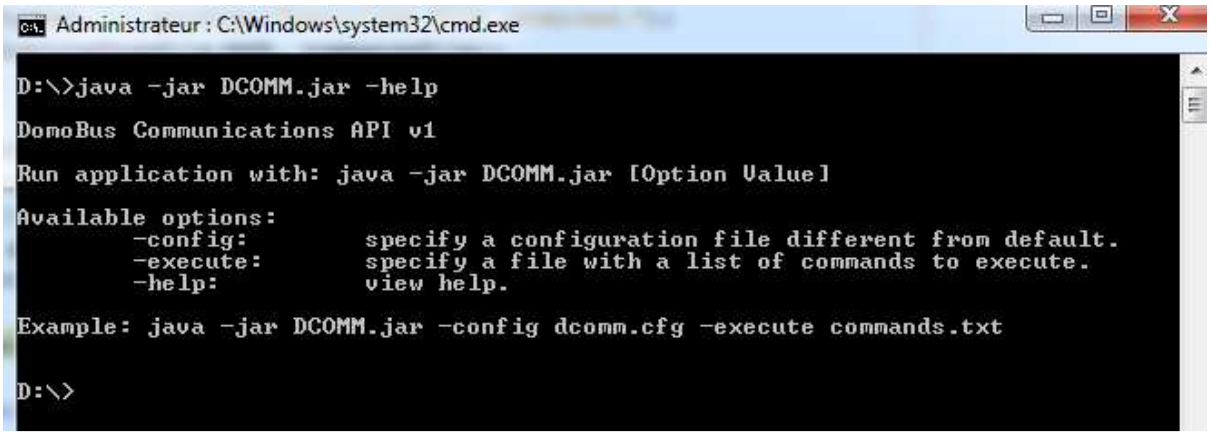


Figure 11 - Test Version: print help

While incorrect options or values would generate the output below.

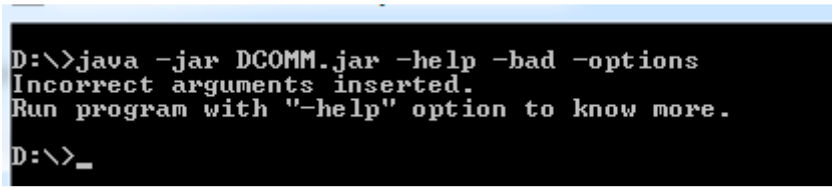


Figure 12 - Test Version: bad arguments

After correctly running the test version, the user should be able see the same options as shown next.

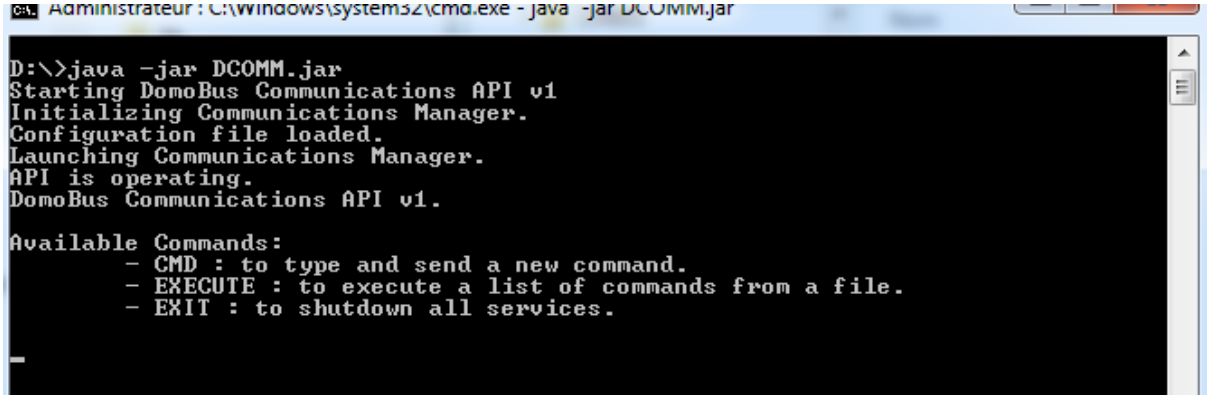


Figure 13 - Test Version: launch

From here we can finally select an option or simply let the Supervisor run within the network and watch interactions that may occur with other copies of the test version. For example, a GET command request should look as follows:

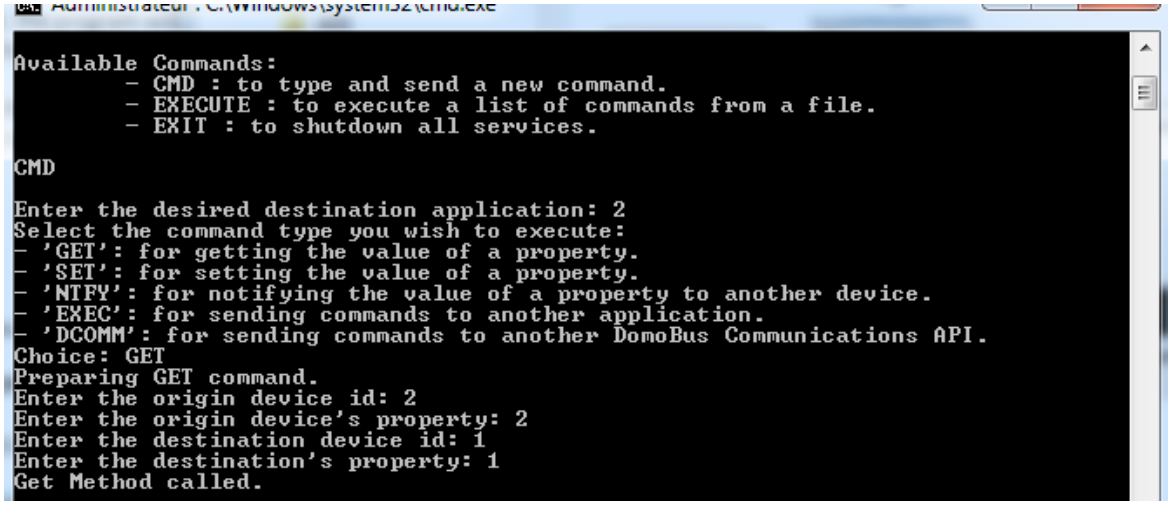


Figure 14 - Test Version: GET command example

Executing a file with valid commands, should generate the following output:

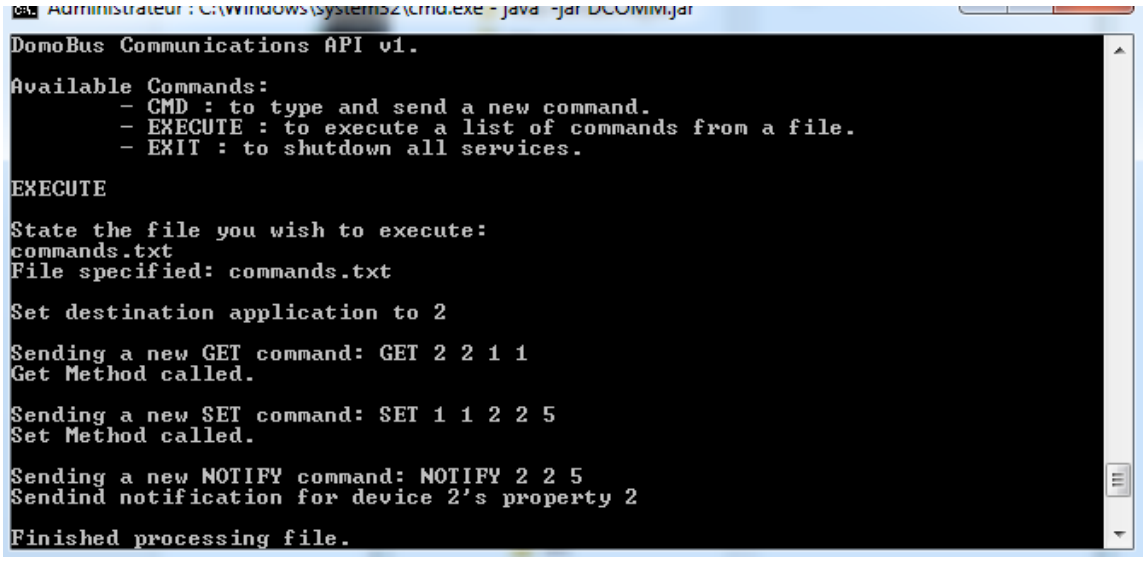


Figure 15 - Test Version: commands file

In this case, we first chose a destination address and then requests a GET command, followed by a SET command and lastly a NOTIFY command.

Finally, shutting down and exiting the DCOMM.jar program:

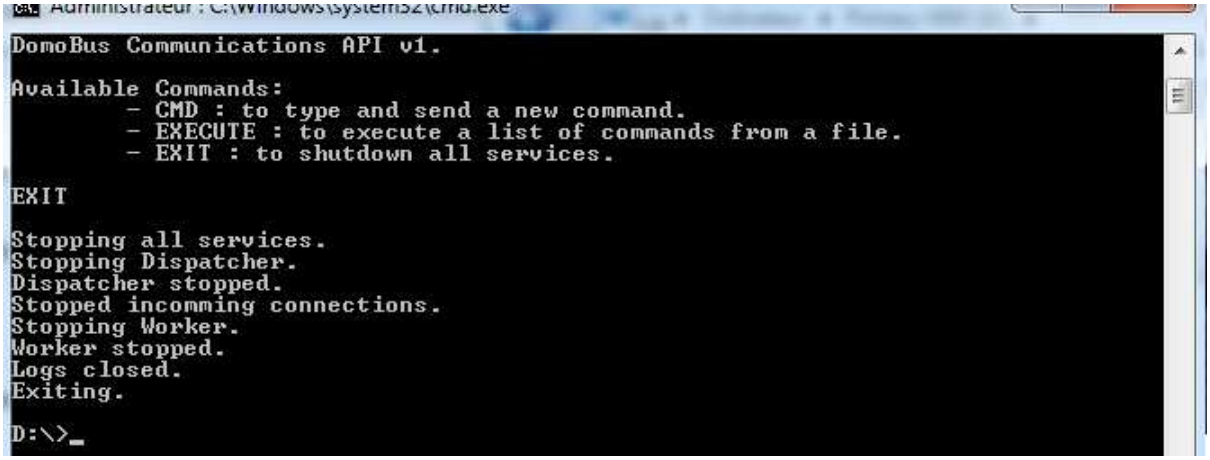


Figure 16 - Test Version: stop operating and exit

Every time some command or a value is incorrectly typed or not supported, for example characters instead of an integer, the current command is aborted and program return to the initial choices.

5. Evaluation

5.1. Setup

The evaluation scheme involves simulating a DomoBus Supervision Level network with three test version's Supervisors running concurrently on the local network of the same machine used for developing the library. And observe how they interact with each other while they execute a multitude of different test commands.

Furthermore, a basic DomoBus Name Server application was created solely for the simulation purposes. It supports Supervisors' registrations and is able to fulfill DNS queries. The DNS, whose full code and JAR file is provided along with this report, makes use of the Logger and Globals classes present in the library. It is intended to provide the address of the Supervisor 3to the Supervisors 1 and 2, as they purposively don't known it before commencing the simulation.

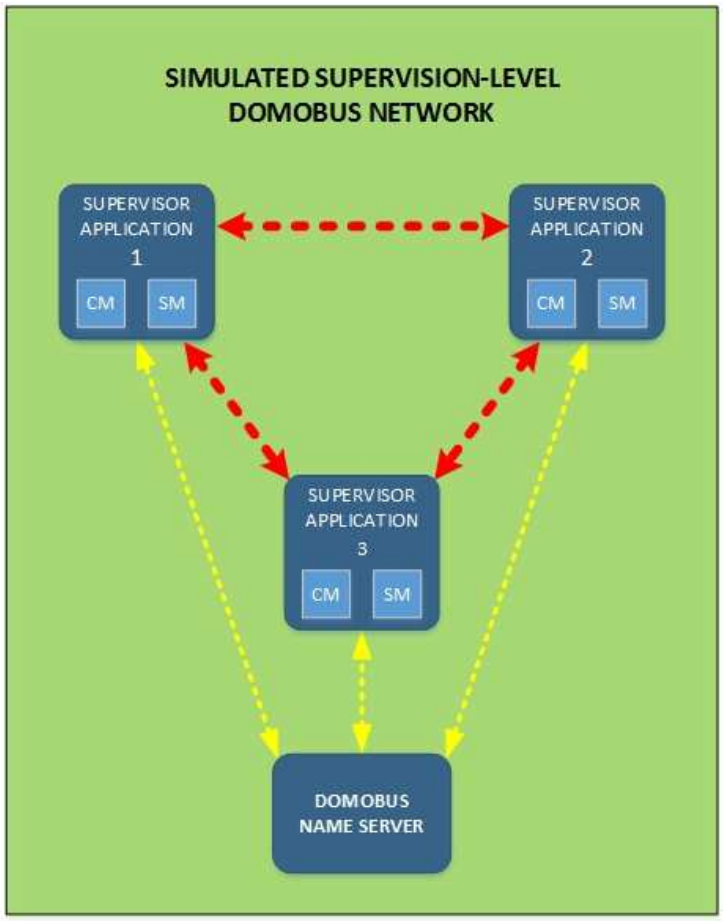


Figure 17 - Network simulation diagram

The only pre-requisite remaining is to create the correct individual configuration file of each simulated Supervisor, all three are detailed in the Annexes. Then, we can proceed to run the test version for each one, making sure the configuration file is located in the same folder or that it is specified during launch.

For this evaluation, we decided to simply name the configuration files differently by appending the Supervisor's ID number and then run the **DCOMM.jar** thrice in separate command-line consoles, each time passing the correct file using the corresponding runtime option, as shown in the figure below.

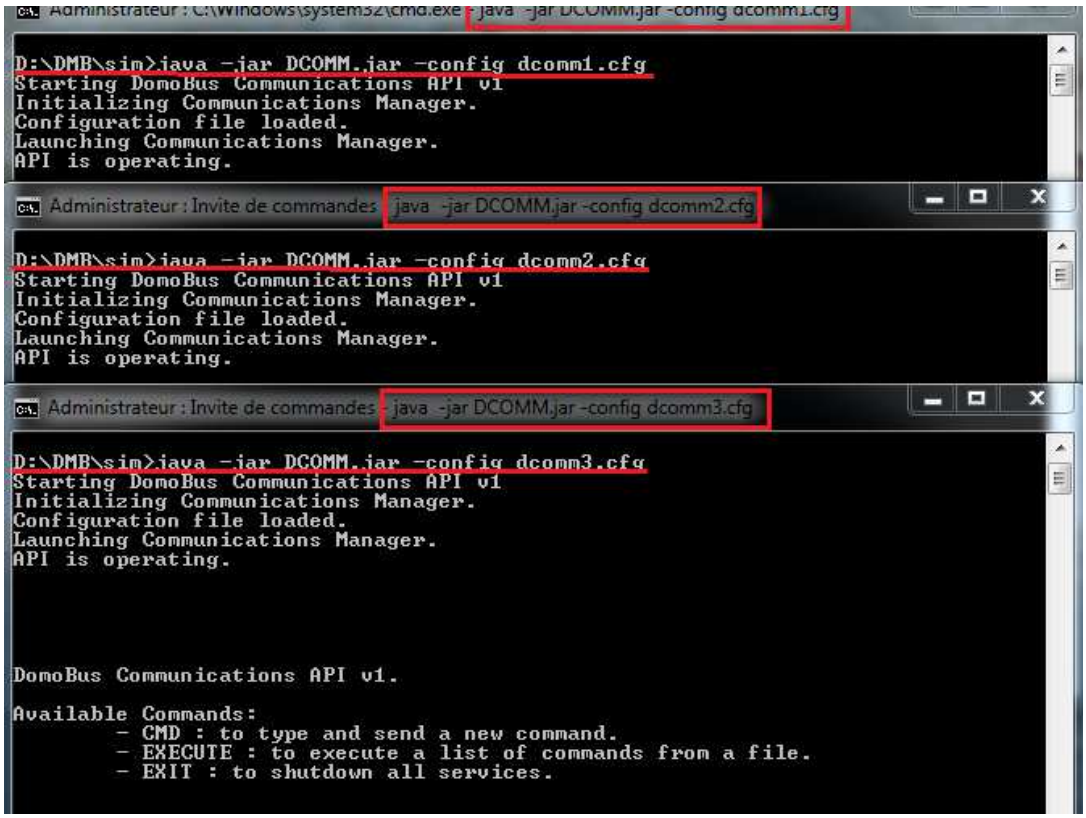


Figure 18 - Launching Test Supervisors

For now, he configuration files defined leave out any prior subscriptions, however this system will be fully tested as well and their resulting state presented.

5.2. Tests

The purpose of the tests is to evaluate all of the functionalities implemented for the communications library. In this sense, the following groups of tests will be formalized and executed:

- DNS Interactions;
- Subscriptions System;
- Basic Operations;
- Synchronous Operations;
- Error Testing.

Note that, as the destination address can change according to the Supervisor being used to execute the commands, a tokenized representation is used and the files are updated as necessary:

- First Peer: refers to the peer Supervisor with the lowest id;
- Second Peer: refers to the peer Supervisor with the highest id.

5.2.1. DNS Interactions

Testing the interactions between the Supervisors and the DomoBus Name Server is done quite simply due to how the library was actually structured, since the first existing DNS service, Supervisors registration, is done by the library's Manager at startup and doesn't require any manual entry.
The second, querying the addresses of unknown Supervisors, is made automatically each time a Supervisor as to either transmit a packet with an unknown destination address or directly receives a new packet from an unknown application address.
Thus, because in our simulation only Supervisor 1 and 2 know each other, we just have to use the Supervisor 3 to send a command to each of them. When processing our commands, the Supervisor 3

will send two **DNS_GET** requests to the DNS to be able to correctly transmit them. While the other two Supervisors will then send their own **DNS_GET** request upon receiving those commands.

Both commands are basic GET operations, with the device addresses and properties set to **1** and they will be typed manually through the Supervisor 3's console.

5.2.2. Subscriptions System

Since there aren't any subscribed applications, we then start by populating each Supervisor's subscriptions list. Afterwards, we will issue some notifications, before and after sending new DCOMM commands to test the unsubscribe functions.
For these tests, all the required commands will be executed from directly from files, whose commented mockups are shown below.

- Part 1: subscribe to properties;

```
SD [First Peer]

#DCOMM_SUBSCRIBE for device 1's property 1
DCOMM 2 011

#DCOMM_SUBSCRIBE for device 1's property 2
DCOMM 2 012

#DCOMM_SUBSCRIBE for device 2's property 2
DCOMM 2 022

SD [Second Peer]

#DCOMM_SUBSCRIBE for device 1's property 1
DCOMM 2 011

#DCOMM_SUBSCRIBE for device 3's property 1
DCOMM 2 031
```

- Part 2: send notifications;

```
NOTIFY 1 1 5
NOTIFY 1 2 5
NOTIFY 2 1 10
NOTIFY 2 2 50
NOTIFY 3 1 9
```

- Part 3: unsubscribe from some or all device-property pairs;

```
SD [First Peer]

#DCOMM_UNSUBSCRIBE for device 1's property 1
DCOMM 3 011

#DCOMM_UNSUBSCRIBE for device 1's property 2
DCOMM 3 012
```

```
SD [Second Peer]

#DCOMM_UNSUBSCRIBE_ALL
DCOMM 4
```

- Part 4: repeat the same notifications sent earlier.

5.2.3. Basic Operations

Here, the remaining supported operations will be tested, GET, SET and EXEC.

```
SD [First Peer]
GET 1 1 2 2
SET 1 1 2 2 5
EXEC 1 012

SD [Second Peer]
GET 1 3 2 2
SET 4 1 2 2 8
EXEC 7 25
```

The sequence of the operations and their values could be different if one wanted, but the result would remain the same considering how the test version was implement since no actual value is kept or the modifications registered, as the Control-Level network is outside the scope of the project and thus not simulated.

Additionally, the first SET command will even generate notifications, because there would still be a subscription for the property 2 of the device 2.

5.2.4. Synchronous Operations

This group of tests will be a repetition of the previous tests. The only difference is actually at the software-level, since, by default, the Manager of the test version only executes the asynchronous methods.
Therefore we must modify the methods, called from the DCOMM main class, from **DComm_send_msg_[OPERATION]** to **DComm_send_sync_msg_[OPERATION]**, and catch the returned variables to display the responses of the commands. Then we need to compile the updated test version and launch once again all three Supervisors to execute the commands file we created before.

5.2.5. Error Testing

The most common errors will be tested. We first revert back to the asynchronous calls and also make some adjustments to the implemented DComm interface. Specifically, we alter the **GET** implementation, setting it to return an error response instead of a simple acknowledgement as previously, and the **EXEC** one, rendering it unsupported and thus will throw the **UnsupportedCommandException**.
Furthermore, we will disconnect the Supervisor 1 and the DomoBus Name Server, and request the other Supervisors to try sending messages to it and an unknown Supervisor 4, whose address they couldn’t resolve in any case.

```
SD 1
GET 2 2 0 0

SD 4
SET 3 2 1 2 4

SD [Second Peer]
GET 1 1 1 1
EXEC 0 50
```

Through those simple tests, we will be able to observe how failed retransmissions, invalid or unsupported commands and the error callback are handled by the library.

5.3. Results

In this section we will display the relevant outputs generated, during the execution of all the established tests, into either the respective consoles or logs files. And even the resulting configuration files. All this information will allow us to analyze and evaluate what was implemented through this project.

To be noted that due to their size, console outputs and logs for the groups 2, 3 and 5 of tests were truncated. But even some repeated events are left as to demonstrate the information obtained through the normal execution of the communications module.

5.3.1. DNS Tests

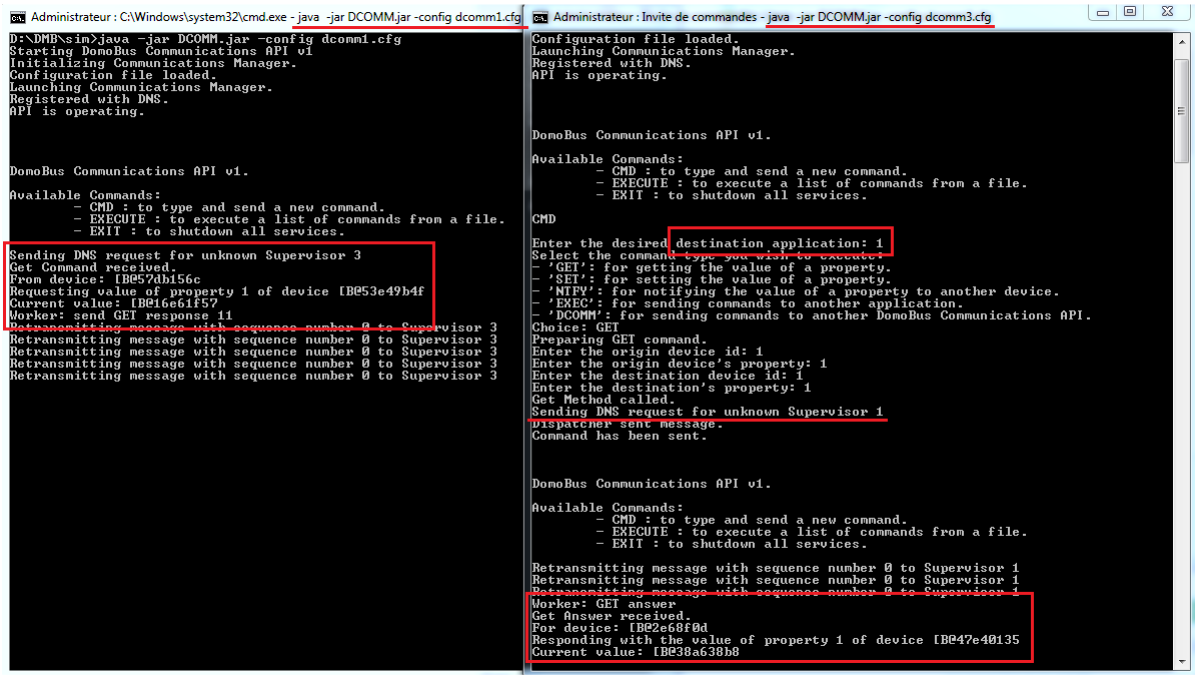


Figure 19 - DNS Tests: Supervisors 1 and 3

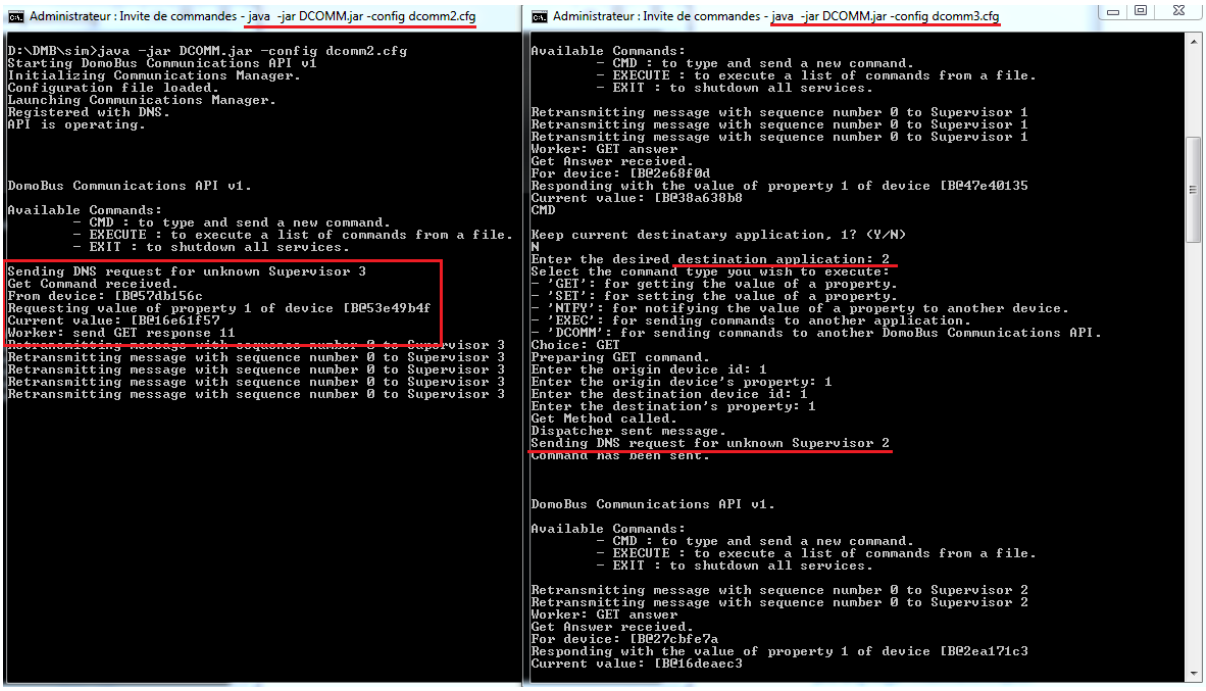


Figure 20 - DNS Tests: Supervisors 2 (left) and 3 (right)

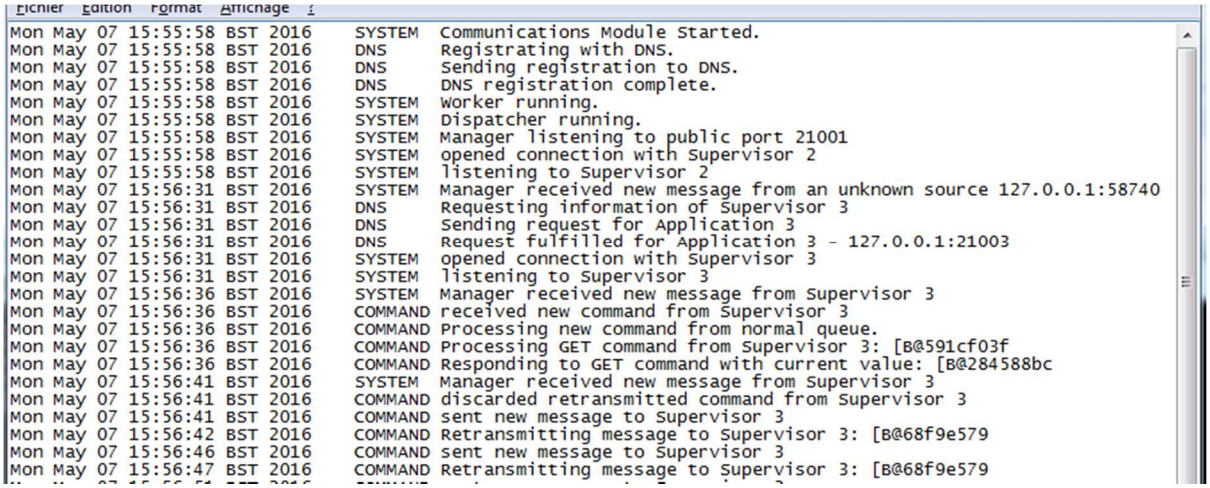


Figure 21 - DNS Tests: Supervisor 1 logs

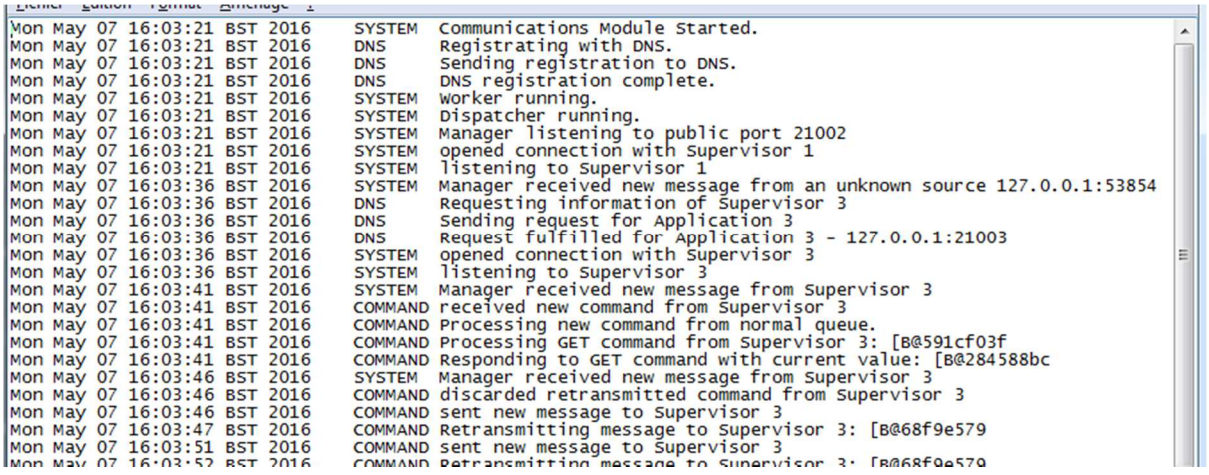


Figure 22 - DNS Tests: Supervisor 2 logs

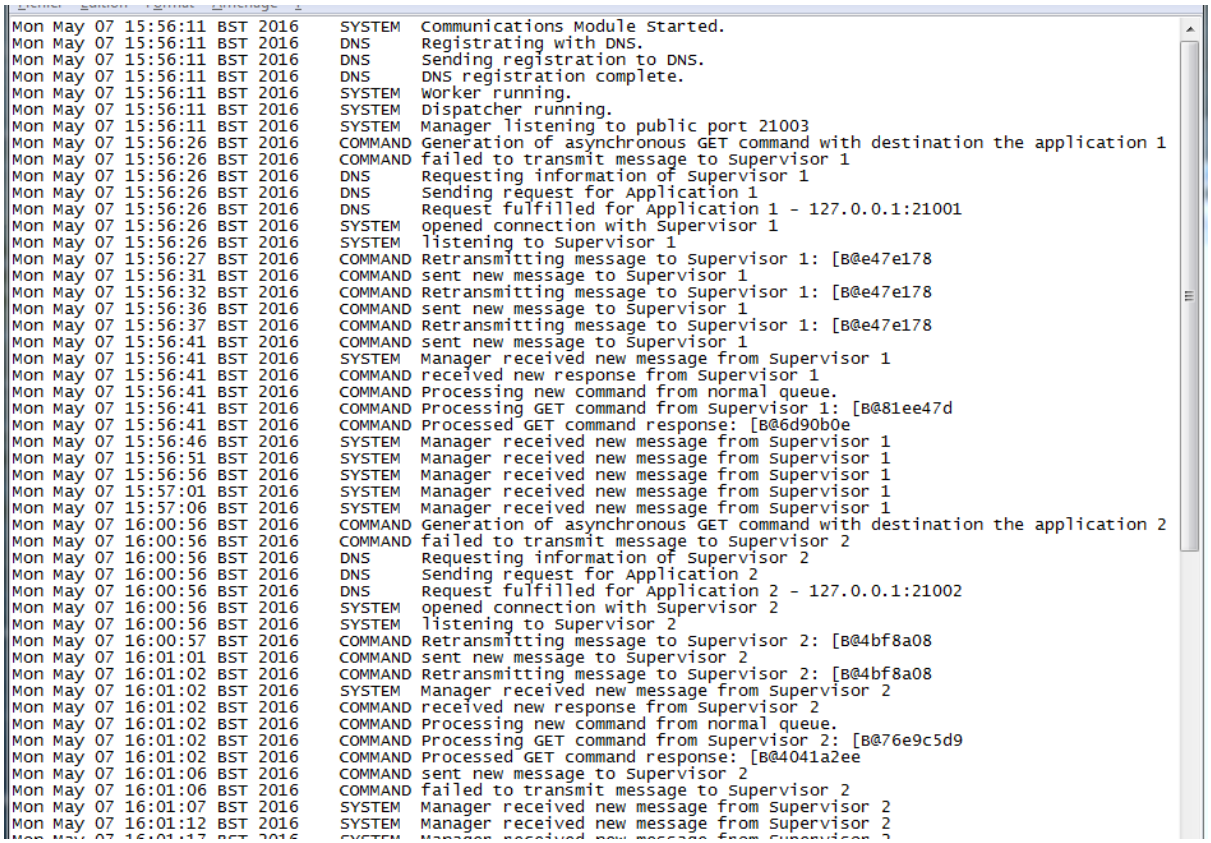


Figure 23 - DNS Tests: Supervisor 3 logs

5.3.2. Subscription Tests

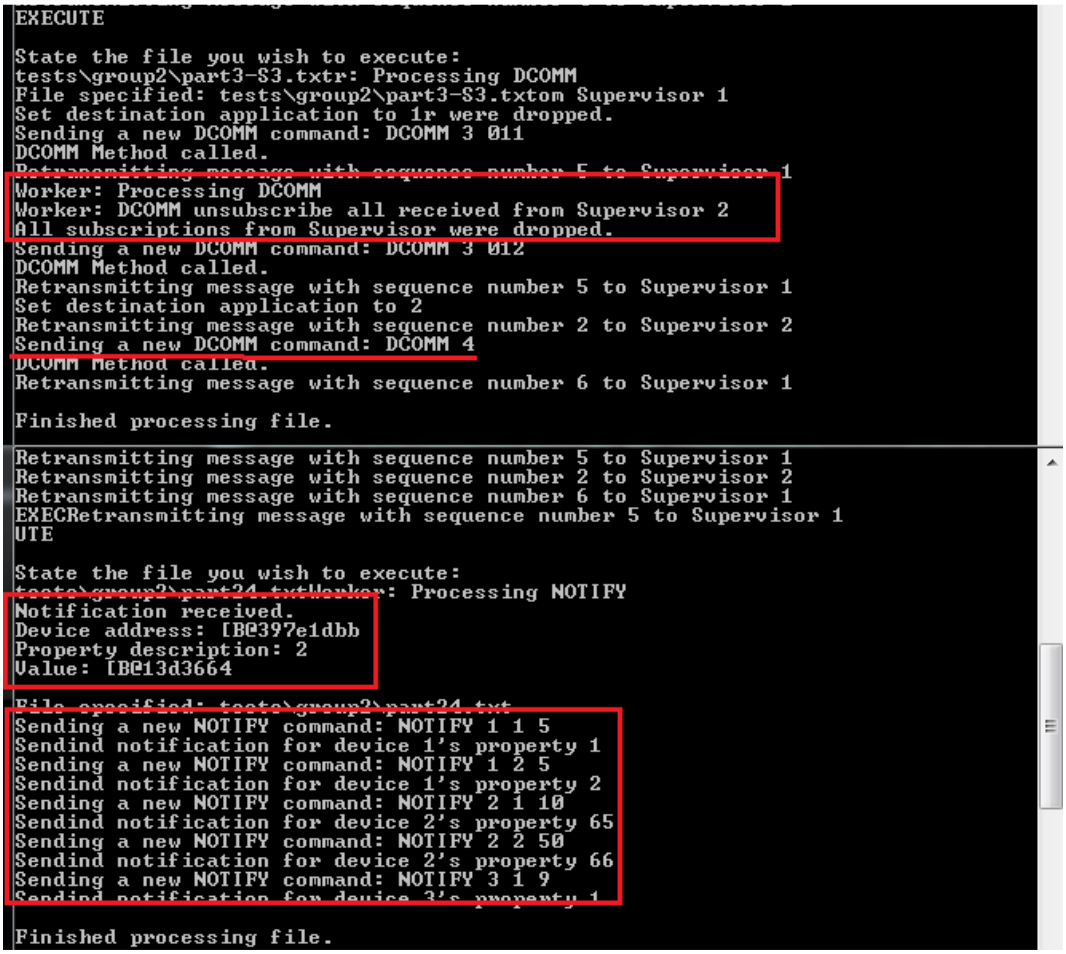


Figure 24 - Subscription Tests: Supervisor 3

```
D:\DMB\sim>java -jar DCOMM.jar -config dcomm1.cfg
Starting DomoBus Communications API v1
Initializing Communications Manager.
Configuration file loaded.
Launching Communications Manager.
API is operating.



DomoBus Communications API v1.

Available Commands:
        - CMD : to type and send a new command.
        - EXECUTE : to execute a list of commands from a file.
        - EXIT : to shutdown all services.

EXECUTE

State the file you wish to execute:
tests\group2\part1-S1.txt
File specified: tests\group2\part1-S1.txt
Set destination application to 2
Sending a new DCOMM command: DCOMM 2 011
DCOMM Method called.
Sending a new DCOMM command: DCOMM 2 012
DCOMM Method called.
Retransmitting message with sequence number 0 to Supervisor 2
Worker: Processing DCOMM
Worker: DCOMM subscribe received from Supervisor 3
New subscription from Supervisor 3 of property 1 from device 1
Sending a new DCOMM command: DCOMM 2 022
DCOMM Method called.
Retransmitting message with sequence number 0 to Supervisor 3
Worker: Processing DCOMM
Worker: DCOMM subscribe received from Supervisor 3
New subscription from Supervisor 3 of property 2 from device 1
Retransmitting message with sequence number 1 to Supervisor 2
Set destination application to 3
Sending a new DCOMM command: DCOMM 2 011
DCOMM Method called.
Retransmitting message with sequence number 0 to Supervisor 2
Worker: Processing DCOMM
Worker: DCOMM subscribe received from Supervisor 3
New subscription from Supervisor 3 of property 2 from device 2
Sending a new DCOMM command: DCOMM 2 031
DCOMM Method called.
Retransmitting message with sequence number 0 to Supervisor 3

Finished processing file.
Retransmitting message with sequence number 1 to Supervisor 3
```

```
Retransmitting message with sequence number 1 to Supervisor 2
EXECUTE

State the file you wish to execute:
Retransmitting message with sequence number 2 to Supervisor 3
tests\group2\part24.txtsmitting message with sequence number 0
File specified: tests\group2\part24.txt
Sending a new NOTIFY command: NOTIFY 1 1 5
Sendind notification for device 1's property 1
Sending a new NOTIFY command: NOTIFY 1 2 5
Sendind notification for device 1's property 2
Retransmitting message with sequence number 0 to Supervisor 3
Worker: Processing NOTIFY
Notification received.
Device address: IB@4c987853
Property description: 1
Value: IB@3122529a
Received NOTIFY acknowledge from Supervisor 3
Sending a new NOTIFY command: NOTIFY 2 1 10
Retransmitting message with sequence number 1 to Supervisor 3
Sendind notification for device 2's property 65
Sending a new NOTIFY command: NOTIFY 2 2 50
Sendind notification for device 2's property 66
Sending a new NOTIFY command: NOTIFY 3 1 9
Sendind notification for device 3's property 1

Finished processing file.

Worker: Processing NOTIFY
Notification received.
Device address: IB@47837239
Property description: 2
Value: IB@48d6c07a
Received NOTIFY acknowledge from Supervisor 3
Retransmitting message with sequence number 2 to Supervisor 2
EXECUTE

State the file you wish to execute:
tests\group2\part3-S1.txt
File specified: tests\group2\part3-S1.txt
Set destination application to 2
Sending a new DCOMM command: DCOMM 3 011
DCOMM Method called.
Sending a new DCOMM command: DCOMM 3 012
DCOMM Method called.
Worker: Processing DCOMM
Received DCOMM acknowledge from Supervisor 2
Set destination application to 3
Sending a new DCOMM command: DCOMM 4
DCOMM Method called.
Worker: Processing DCOMM
Received DCOMM acknowledge from Supervisor 2

Finished processing file.

Retransmitting message with sequence number 5 to Supervisor 2
EXECUTE

State the file you wish to execute:
tests\group2\part24.txt
File specified: tests\group2\part24.txt
Sending a new NOTIFY command: NOTIFY 1 1 5
Sendind notification for device 1's property 1
Sending a new NOTIFY command: NOTIFY 1 2 5
Sendind notification for device 1's property 2
Retransmitting message with sequence number 6 to Supervisor 3
Sending a new NOTIFY command: NOTIFY 2 1 10
Sendind notification for device 2's property 65
Sending a new NOTIFY command: NOTIFY 2 2 50
Sendind notification for device 2's property 66
Sending a new NOTIFY command: NOTIFY 3 1 9
Sendind notification for device 3's property 1

Finished processing file.
```

Figure 25 - Subscription Tests: Supervisor 1

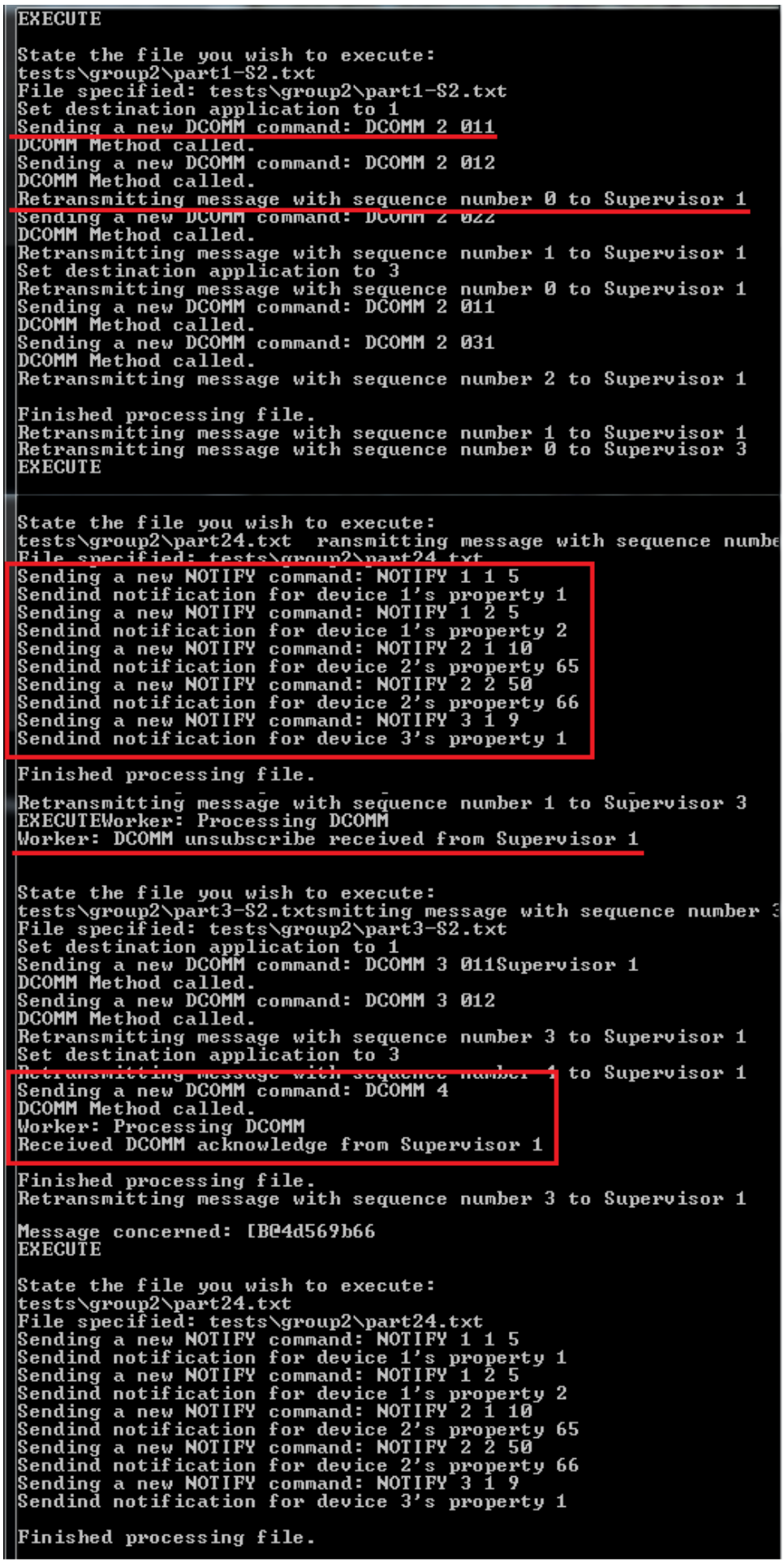


Figure 26 - Subscription Tests: Supervisor 2

```
Fichier  Edition  Format  Affichage  ?
Mon May 07 21:51:43 BST 2016    SYSTEM  Listening to Supervisor 2
Mon May 07 21:51:56 BST 2016    COMMAND Executing commands file: tests\group2\part1-S1.txt
Mon May 07 21:51:56 BST 2016    COMMAND Generation of asynchronous DCOMM command with destination the application 2
Mon May 07 21:51:58 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 21:51:58 BST 2016    COMMAND Generation of asynchronous DCOMM command with destination the application 2
Mon May 07 21:51:59 BST 2016    SYSTEM  Manager received new message from Supervisor 2
Mon May 07 21:51:59 BST 2016    COMMAND discarded duplicate command from Supervisor 2
Mon May 07 21:51:59 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@3af573a
Mon May 07 21:52:01 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 21:52:01 BST 2016    COMMAND received new command from Supervisor 3
Mon May 07 21:52:01 BST 2016    COMMAND Processing new command from priority queue.
Mon May 07 21:52:01 BST 2016    COMMAND Processing DCOMM command from Supervisor 3: [B@634d6f3e
Mon May 07 21:52:01 BST 2016    COMMAND Added subscription for Supervisor 3 of property 1 from device 1
Mon May 07 21:52:01 BST 2016    COMMAND Responding to DCOMM command with an acknowledgement: [B@6d8a8d27
Mon May 07 21:52:03 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 21:52:03 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 21:52:03 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 21:52:03 BST 2016    COMMAND Generation of asynchronous DCOMM command with destination the application 2
Mon May 07 21:52:04 BST 2016    SYSTEM  Manager received new message from Supervisor 2
Mon May 07 21:52:04 BST 2016    COMMAND discarded duplicate command from Supervisor 2
Mon May 07 21:52:04 BST 2016    SYSTEM  Manager received new message from Supervisor 2
Mon May 07 21:52:04 BST 2016    COMMAND discarded retransmitted command from Supervisor 2
Mon May 07 21:52:04 BST 2016    COMMAND Retransmitting message to Supervisor 3: [B@d68a93f
Mon May 07 21:52:06 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 21:52:06 BST 2016    COMMAND received new command from Supervisor 3
Mon May 07 21:52:06 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 21:52:06 BST 2016    COMMAND discarded retransmitted command from Supervisor 3
Mon May 07 21:52:06 BST 2016    COMMAND Processing new command from priority queue.
Mon May 07 21:52:06 BST 2016    COMMAND Processing DCOMM command from Supervisor 3: [B@69b5044e
Mon May 07 21:52:06 BST 2016    COMMAND Added subscription for Supervisor 3 of property 2 from device 1
Mon May 07 21:52:06 BST 2016    COMMAND Responding to DCOMM command with an acknowledgement: [B@2cb7871c
Mon May 07 21:52:11 BST 2016    COMMAND discarded retransmitted command from Supervisor 3
Mon May 07 21:52:11 BST 2016    COMMAND Processing new command from priority queue.
Mon May 07 21:52:11 BST 2016    COMMAND Processing DCOMM command from Supervisor 3: [B@240b292b
Mon May 07 21:52:11 BST 2016    COMMAND Added subscription for Supervisor 3 of property 2 from device 2
Mon May 07 21:52:11 BST 2016    COMMAND Responding to DCOMM command with an acknowledgement: [B@1dd16b38
Mon May 07 21:52:13 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 21:52:13 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 21:52:13 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 21:52:13 BST 2016    COMMAND Generation of asynchronous DCOMM command with destination the application 3
Mon May 07 21:52:13 BST 2016    COMMAND Retransmitting message to Supervisor 3: [B@d68a93f
Mon May 07 21:52:14 BST 2016    SYSTEM  Manager received new message from Supervisor 2
Mon May 07 21:52:14 BST 2016    COMMAND discarded retransmitted command from Supervisor 2
Mon May 07 21:52:16 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 21:52:16 BST 2016    COMMAND discarded retransmitted command from Supervisor 3
Mon May 07 21:52:18 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 21:52:18 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 21:52:18 BST 2016    COMMAND Retransmitting message to Supervisor 3: [B@4c55a304
Mon May 07 21:52:18 BST 2016    COMMAND Successfully finished execution of commands file: tests\group2\part1-S1.txt
Mon May 07 21:53:38 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@3af573a
Mon May 07 21:53:39 BST 2016    COMMAND Executing commands file: tests\group2\part24.txt
Mon May 07 21:53:39 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 1 of device 1
Mon May 07 21:53:39 BST 2016    COMMAND Notifying Supervisor 3 the current value of property 1 of device 1: [B@266474c2
Mon May 07 21:53:39 BST 2016    SYSTEM  Manager received new message from Supervisor 2
Mon May 07 21:53:39 BST 2016    COMMAND discarded retransmitted command from Supervisor 2
Mon May 07 21:53:43 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 21:53:43 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 21:53:43 BST 2016    COMMAND Retransmitting message to Supervisor 3: [B@d68a93f
Mon May 07 21:53:43 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 2 of device 1
Mon May 07 21:53:43 BST 2016    COMMAND Notifying Supervisor 3 the current value of property 2 of device 1: [B@6f94fa3e
Mon May 07 21:53:46 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 21:53:46 BST 2016    COMMAND received new response from Supervisor 3
Mon May 07 21:53:46 BST 2016    COMMAND Processing new command from priority queue.
Mon May 07 21:53:46 BST 2016    COMMAND Received NOTIFY command from Supervisor 3: [B@5c879bb8
Mon May 07 21:53:46 BST 2016    COMMAND Received NOTIFY acknowledge from Supervisor 3
Mon May 07 21:53:48 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 21:53:48 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 21:53:48 BST 2016    COMMAND Retransmitting message to Supervisor 3: [B@4c55a304
Mon May 07 21:53:48 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 65 of device 2
Mon May 07 21:53:48 BST 2016    COMMAND No subscriptions for property 65 of device 2
Mon May 07 21:53:48 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 66 of device 2
Mon May 07 21:53:48 BST 2016    COMMAND No subscriptions for property 66 of device 2
Mon May 07 21:53:48 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 1 of device 3
Mon May 07 21:53:48 BST 2016    COMMAND No subscriptions for property 1 of device 3
Mon May 07 21:53:48 BST 2016    COMMAND Successfully finished execution of commands file: tests\group2\part24.txt
Mon May 07 21:53:51 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 21:53:51 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 21:53:51 BST 2016    COMMAND received new response from Supervisor 3
Mon May 07 21:53:51 BST 2016    COMMAND Processing new command from priority queue.
Mon May 07 21:53:51 BST 2016    COMMAND Received NOTIFY command from Supervisor 3: [B@27b9772d
Mon May 07 21:53:51 BST 2016    COMMAND Received NOTIFY acknowledge from Supervisor 3
Mon May 07 21:53:53 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 21:53:53 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@353a2b33
Mon May 07 21:55:48 BST 2016    COMMAND Executing commands file: tests\group2\part24.txt
Mon May 07 21:55:48 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 1 of device 1
Mon May 07 21:55:48 BST 2016    COMMAND Notifying Supervisor 3 the current value of property 1 of device 1: [B@5e481248
Mon May 07 21:55:53 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 21:55:53 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 2 of device 1
Mon May 07 21:55:53 BST 2016    COMMAND Notifying Supervisor 3 the current value of property 2 of device 1: [B@66d3c617
Mon May 07 21:55:54 BST 2016    COMMAND Retransmitting message to Supervisor 3: [B@487d8944
Mon May 07 21:55:56 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 21:55:56 BST 2016    COMMAND discarded retransmitted command from Supervisor 3
Mon May 07 21:55:58 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 21:55:58 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 21:55:58 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 65 of device 2
Mon May 07 21:55:58 BST 2016    COMMAND No subscriptions for property 65 of device 2
Mon May 07 21:55:58 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 66 of device 2
Mon May 07 21:55:58 BST 2016    COMMAND No subscriptions for property 66 of device 2
Mon May 07 21:55:58 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 1 of device 3
Mon May 07 21:55:58 BST 2016    COMMAND No subscriptions for property 1 of device 3
Mon May 07 21:55:58 BST 2016    COMMAND Successfully finished execution of commands file: tests\group2\part24.txt
Mon May 07 21:55:59 BST 2016    COMMAND Retransmitting message to Supervisor 3: [B@2f26a12c
```

Figure 27 - Subscription Tests: Supervisor 1 logs

```
Mon May 07 21:51:44 BST 2016    SYSTEM  Manager listening to public port 21002
Mon May 07 21:51:44 BST 2016    SYSTEM  opened connection with Supervisor 3
Mon May 07 21:51:44 BST 2016    SYSTEM  listening to Supervisor 3
Mon May 07 21:51:44 BST 2016    SYSTEM  opened connection with Supervisor 1
Mon May 07 21:51:44 BST 2016    SYSTEM  listening to Supervisor 1
Mon May 07 21:51:57 BST 2016    COMMAND Executing commands file: tests\group2\part1-S2.txt
Mon May 07 21:51:57 BST 2016    COMMAND Generation of asynchronous DCOMM command with destination the application 1
Mon May 07 21:51:58 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:51:59 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:51:59 BST 2016    COMMAND Generation of asynchronous DCOMM command with destination the application 1
Mon May 07 21:52:00 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@3af573a
Mon May 07 21:52:03 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:52:03 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:52:19 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:52:19 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 21:52:19 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@d68a93f
Mon May 07 21:52:19 BST 2016    COMMAND Successfully finished execution of commands file: tests\group2\part1-S2.txt
Mon May 07 21:52:21 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 21:52:21 BST 2016    COMMAND discarded duplicate command from Supervisor 3
Mon May 07 21:52:21 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 21:52:28 BST 2016    COMMAND discarded retransmitted command from Supervisor 1
Mon May 07 21:52:29 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:52:29 BST 2016    COMMAND Retransmitting message to Supervisor 3: [B@4c55a304
Mon May 07 21:52:31 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 21:52:31 BST 2016    COMMAND discarded retransmitted command from Supervisor 3
Mon May 07 21:52:33 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:53:43 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:53:44 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 21:53:44 BST 2016    COMMAND Retransmitting message to Supervisor 3: [B@353a2b33
Mon May 07 21:53:48 BST 2016    COMMAND Executing commands file: tests\group2\part24.txt
Mon May 07 21:53:48 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 1 of device 1
Mon May 07 21:53:48 BST 2016    COMMAND No subscriptions for property 1 of device 1
Mon May 07 21:53:48 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 2 of device 1
Mon May 07 21:53:48 BST 2016    COMMAND No subscriptions for property 2 of device 1
Mon May 07 21:53:48 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 65 of device 2
Mon May 07 21:53:48 BST 2016    COMMAND No subscriptions for property 65 of device 2
Mon May 07 21:53:48 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 66 of device 2
Mon May 07 21:53:48 BST 2016    COMMAND No subscriptions for property 66 of device 2
Mon May 07 21:53:48 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 1 of device 3
Mon May 07 21:53:48 BST 2016    COMMAND No subscriptions for property 1 of device 3
Mon May 07 21:53:48 BST 2016    COMMAND Successfully finished execution of commands file: tests\group2\part24.txt
Mon May 07 21:53:49 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 21:53:49 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@9573c4a
Mon May 07 21:53:54 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:53:54 BST 2016    COMMAND Retransmitting message to Supervisor 3: [B@4c55a304
Mon May 07 21:53:58 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:53:59 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 21:53:59 BST 2016    COMMAND Retransmitting message to Supervisor 3: [B@353a2b33
Mon May 07 21:54:53 BST 2016    COMMAND Processing new command from priority queue.
Mon May 07 21:54:53 BST 2016    COMMAND Processing DCOMM command from Supervisor 1: [B@634d6f3e
Mon May 07 21:54:53 BST 2016    COMMAND Responding to DCOMM command with an acknowledgement: [B@6d8a8d27
Mon May 07 21:54:54 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:54:55 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@7dd34dfa
Mon May 07 21:54:58 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:54:58 BST 2016    COMMAND received new command from Supervisor 1
Mon May 07 21:54:58 BST 2016    COMMAND Processing new command from priority queue.
Mon May 07 21:54:58 BST 2016    COMMAND Processing DCOMM command from Supervisor 1: [B@69b5044e
Mon May 07 21:54:58 BST 2016    COMMAND Responding to DCOMM command with an acknowledgement: [B@2cb7871c
Mon May 07 21:54:58 BST 2016    COMMAND Executing commands file: tests\group2\part3-S2.txt
Mon May 07 21:54:58 BST 2016    COMMAND Generation of asynchronous DCOMM command with destination the application 1
Mon May 07 21:54:59 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:54:59 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:54:59 BST 2016    COMMAND Generation of asynchronous DCOMM command with destination the application 1
Mon May 07 21:55:00 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@7dd34dfa
Mon May 07 21:55:04 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:55:04 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:55:04 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@4d569b66
Mon May 07 21:55:04 BST 2016    COMMAND Generation of asynchronous DCOMM command with destination the application 3
Mon May 07 21:55:08 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:55:08 BST 2016    COMMAND received new response from Supervisor 1
Mon May 07 21:55:08 BST 2016    COMMAND Processing new command from priority queue.
Mon May 07 21:55:08 BST 2016    COMMAND Processing DCOMM command from Supervisor 1: [B@240b292b
Mon May 07 21:55:08 BST 2016    COMMAND Received DCOMM acknowledge from Supervisor 1
Mon May 07 21:55:09 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 21:55:09 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:55:09 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@7dd34dfa
Mon May 07 21:55:09 BST 2016    COMMAND Successfully finished execution of commands file: tests\group2\part3-S2.txt
Mon May 07 21:55:11 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 21:55:11 BST 2016    COMMAND received new response from Supervisor 3
Mon May 07 21:55:11 BST 2016    COMMAND Processing new command from priority queue.
Mon May 07 21:55:13 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:55:14 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:55:34 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:55:34 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@4d569b66
Mon May 07 21:55:36 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 21:55:39 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:55:40 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@4d569b66
Mon May 07 21:55:44 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:55:44 BST 2016    ERROR   TransmissionFailed      The message failed to be transmitted to its destination and h
Mon May 07 21:55:51 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 21:55:54 BST 2016    COMMAND Executing commands file: tests\group2\part24.txt
Mon May 07 21:55:54 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 1 of device 1
Mon May 07 21:55:54 BST 2016    COMMAND No subscriptions for property 1 of device 1
Mon May 07 21:55:54 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 2 of device 1
Mon May 07 21:55:54 BST 2016    COMMAND No subscriptions for property 2 of device 1
Mon May 07 21:55:54 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 65 of device 2
Mon May 07 21:55:54 BST 2016    COMMAND No subscriptions for property 65 of device 2
Mon May 07 21:55:54 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 66 of device 2
Mon May 07 21:55:54 BST 2016    COMMAND No subscriptions for property 66 of device 2
Mon May 07 21:55:54 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 1 of device 3
```

Figure 28 - Subscription Tests: Supervisor 2 logs

```
Mon May 07 21:51:46 BST 2016    SYSTEM  Manager listening to public port 21003
Mon May 07 21:51:46 BST 2016    SYSTEM  opened connection with Supervisor 1
Mon May 07 21:51:46 BST 2016    SYSTEM  listening to Supervisor 1
Mon May 07 21:51:57 BST 2016    COMMAND Executing commands file: tests\group2\part1-S3.txt
Mon May 07 21:51:57 BST 2016    COMMAND Generation of asynchronous DCOMM command with destination the application 1
Mon May 07 21:52:01 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:52:01 BST 2016    COMMAND Generation of asynchronous DCOMM command with destination the application 1
Mon May 07 21:52:02 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@3af573a
Mon May 07 21:52:03 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:52:03 BST 2016    COMMAND received new response from Supervisor 1
Mon May 07 21:52:03 BST 2016    COMMAND Processing new command from priority queue.
Mon May 07 21:52:03 BST 2016    COMMAND Processing DCOMM command from Supervisor 1: [B@634d6f3e
Mon May 07 21:52:03 BST 2016    COMMAND Received DCOMM acknowledge from Supervisor 1
Mon May 07 21:52:06 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:52:06 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:52:06 BST 2016    COMMAND failed to transmit message to Supervisor 1
Mon May 07 21:52:06 BST 2016    COMMAND Generation of asynchronous DCOMM command with destination the application 1
Mon May 07 21:52:07 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@d68a93f
Mon May 07 21:52:08 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:52:08 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:52:08 BST 2016    COMMAND received new response from Supervisor 1
Mon May 07 21:52:08 BST 2016    COMMAND Processing new command from priority queue.
Mon May 07 21:52:08 BST 2016    COMMAND Processing DCOMM command from Supervisor 1: [B@6d8a8d27
Mon May 07 21:52:08 BST 2016    COMMAND Received DCOMM acknowledge from Supervisor 1
Mon May 07 21:52:11 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:52:11 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:52:11 BST 2016    COMMAND failed to transmit message to Supervisor 1
Mon May 07 21:52:11 BST 2016    COMMAND Generation of asynchronous DCOMM command with destination the application 2
Mon May 07 21:52:12 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@9573c4a
Mon May 07 21:52:13 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:52:13 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:52:13 BST 2016    COMMAND received new response from Supervisor 1
Mon May 07 21:52:13 BST 2016    COMMAND Processing new command from priority queue.
Mon May 07 21:52:13 BST 2016    COMMAND Processing DCOMM command from Supervisor 1: [B@69b5044e
Mon May 07 21:52:13 BST 2016    COMMAND Received DCOMM acknowledge from Supervisor 1
Mon May 07 21:52:14 BST 2016    SYSTEM  Manager received new message from Supervisor 2
Mon May 07 21:52:16 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:52:16 BST 2016    COMMAND failed to transmit message to Supervisor 1
Mon May 07 21:52:16 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 21:52:16 BST 2016    COMMAND Generation of asynchronous DCOMM command with destination the application 2
Mon May 07 21:52:17 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@4c55a304
Mon May 07 21:52:18 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:52:18 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:52:19 BST 2016    SYSTEM  Manager received new message from Supervisor 2
Mon May 07 21:52:21 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 21:52:21 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 21:52:21 BST 2016    COMMAND Successfully finished execution of commands file: tests\group2\part1-S3.txt
Mon May 07 21:52:22 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@353a2b33
Mon May 07 21:52:23 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:52:26 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 21:52:26 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@4c55a304
Mon May 07 21:52:31 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 21:52:31 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@353a2b33
Mon May 07 21:52:34 BST 2016    SYSTEM  Manager received new message from Supervisor 2
Mon May 07 21:52:34 BST 2016    COMMAND discarded retransmitted command from Supervisor 2
Mon May 07 21:53:43 BST 2016    COMMAND Received NOTIFY command from Supervisor 1: [B@2cb7871c
Mon May 07 21:53:43 BST 2016    COMMAND Acknowledging NOTIFY command from Supervisor 1: [B@5c879bb8
Mon May 07 21:53:44 BST 2016    SYSTEM  Manager received new message from Supervisor 2
Mon May 07 21:53:46 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:53:47 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@7dd34dfa
Mon May 07 21:53:48 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:53:48 BST 2016    COMMAND received new command from Supervisor 1
Mon May 07 21:53:48 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:53:48 BST 2016    COMMAND Processing new command from priority queue.
Mon May 07 21:53:48 BST 2016    COMMAND Received NOTIFY command from Supervisor 1: [B@4c987853
Mon May 07 21:53:48 BST 2016    COMMAND Acknowledging NOTIFY command from Supervisor 1: [B@47837239
Mon May 07 21:53:49 BST 2016    SYSTEM  Manager received new message from Supervisor 2
Mon May 07 21:53:51 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:53:51 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:53:52 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@7dd34dfa
Mon May 07 21:53:53 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:53:56 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:53:56 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@4d569b66
Mon May 07 21:53:57 BST 2016    COMMAND Executing commands file: tests\group2\part24.txt
Mon May 07 21:53:57 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 1 of device 1
Mon May 07 21:53:57 BST 2016    COMMAND No subscriptions for property 1 of device 1
Mon May 07 21:53:57 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 2 of device 1
Mon May 07 21:53:57 BST 2016    COMMAND No subscriptions for property 2 of device 1
Mon May 07 21:53:57 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 65 of device 2
Mon May 07 21:53:57 BST 2016    COMMAND No subscriptions for property 65 of device 2
Mon May 07 21:53:57 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 66 of device 2
Mon May 07 21:53:57 BST 2016    COMMAND No subscriptions for property 66 of device 2
Mon May 07 21:53:57 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 1 of device 3
Mon May 07 21:53:57 BST 2016    COMMAND No subscriptions for property 1 of device 3
Mon May 07 21:53:57 BST 2016    COMMAND Successfully finished execution of commands file: tests\group2\part24.txt
Mon May 07 21:53:59 BST 2016    SYSTEM  Manager received new message from Supervisor 2
Mon May 07 21:55:03 BST 2016    COMMAND Processing new command from priority queue.
Mon May 07 21:55:03 BST 2016    COMMAND Processing DCOMM command from Supervisor 1: [B@48d6c07a
Mon May 07 21:55:03 BST 2016    COMMAND Removing all subscriptions of Supervisor 1
Mon May 07 21:55:03 BST 2016    COMMAND Responding to DCOMM command with an acknowledgement: [B@436ef878
Mon May 07 21:55:06 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:55:07 BST 2016    COMMAND Executing commands file: tests\group2\part3-S3.txt
Mon May 07 21:55:07 BST 2016    COMMAND Generation of asynchronous DCOMM command with destination the application 1
Mon May 07 21:55:07 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@2c7a5c4b
Mon May 07 21:55:08 BST 2016    SYSTEM  Manager received new message from Supervisor 1
Mon May 07 21:55:08 BST 2016    COMMAND discarded retransmitted command from Supervisor 1
Mon May 07 21:55:09 BST 2016    SYSTEM  Manager received new message from Supervisor 2
Mon May 07 21:55:09 BST 2016    COMMAND received new command from Supervisor 2
Mon May 07 21:55:09 BST 2016    COMMAND Processing new command from priority queue.
Mon May 07 21:55:09 BST 2016    COMMAND Processing DCOMM command from Supervisor 2: [B@2f82ef27
Mon May 07 21:55:09 BST 2016    COMMAND Removing all subscriptions of Supervisor 2
Mon May 07 21:55:09 BST 2016    COMMAND Responding to DCOMM command with an acknowledgement: [B@73cc5254
Mon May 07 21:55:11 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:55:11 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 21:55:11 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 21:55:11 BST 2016    COMMAND Generation of asynchronous DCOMM command with destination the application 1
Mon May 07 21:55:12 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@487d8944
Mon May 07 21:55:58 BST 2016    COMMAND Acknowledging NOTIFY command from Supervisor 1: [B@10f28ae5
Mon May 07 21:55:58 BST 2016    COMMAND Executing commands file: tests\group2\part24.txt
Mon May 07 21:55:58 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 1 of device 1
Mon May 07 21:55:58 BST 2016    COMMAND No subscriptions for property 1 of device 1
Mon May 07 21:55:58 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 2 of device 1
Mon May 07 21:55:58 BST 2016    COMMAND No subscriptions for property 2 of device 1
Mon May 07 21:55:58 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 65 of device 2
Mon May 07 21:55:58 BST 2016    COMMAND No subscriptions for property 65 of device 2
Mon May 07 21:55:58 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 66 of device 2
Mon May 07 21:55:58 BST 2016    COMMAND No subscriptions for property 66 of device 2
Mon May 07 21:55:58 BST 2016    COMMAND Generation of asynchronous NOTIFY commands for property 1 of device 3
Mon May 07 21:55:58 BST 2016    COMMAND No subscriptions for property 1 of device 3
Mon May 07 21:55:58 BST 2016    COMMAND Successfully finished execution of commands file: tests\group2\part24.txt
Mon May 07 21:56:01 BST 2016    COMMAND sent new message to Supervisor 1
```

Figure 29 - Subscription Tests: Supervisor 3 logs

5.3.3. Asynchronous Tests

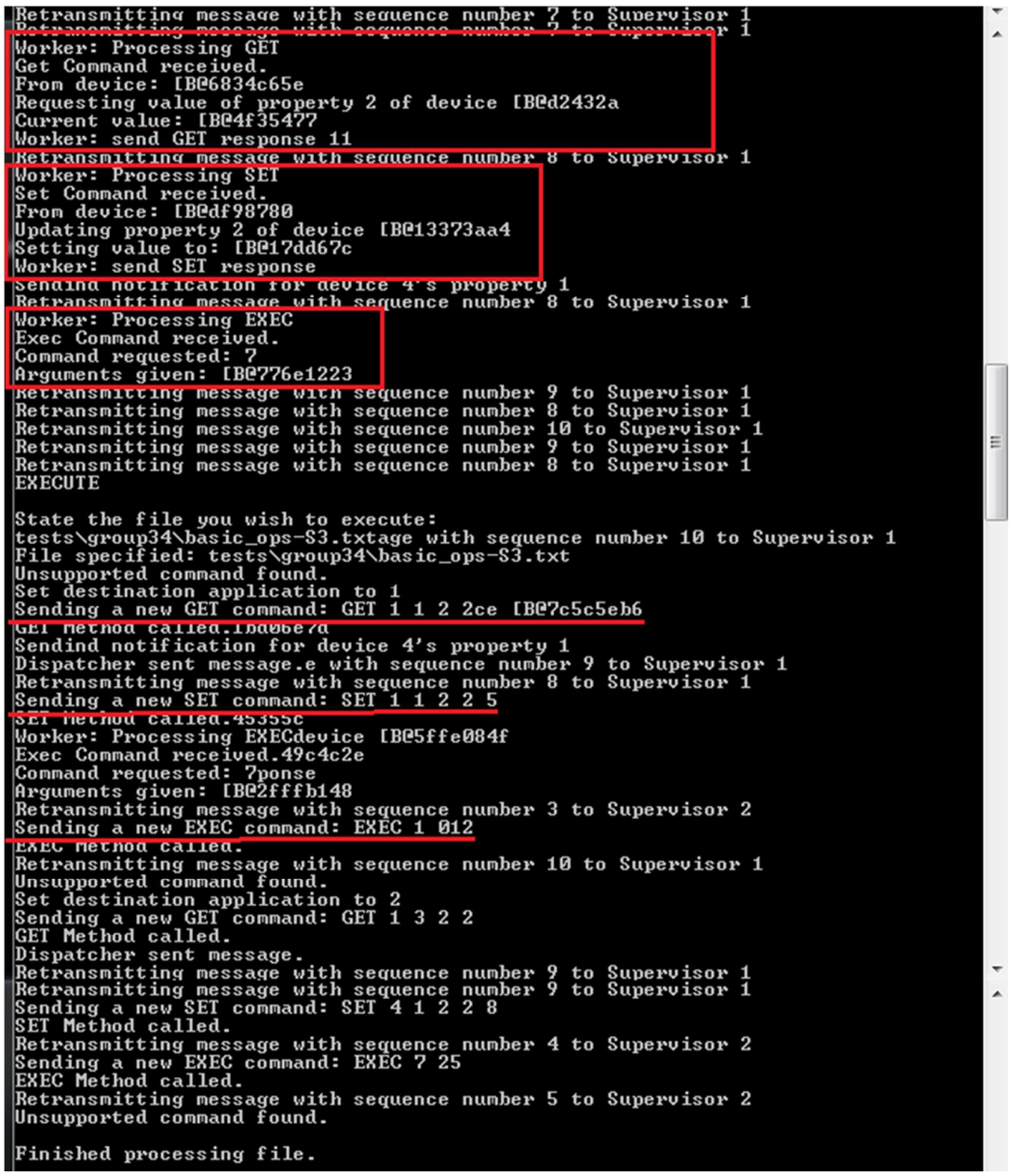


Figure 30 - Asynchronous Tests: Supervisor 3

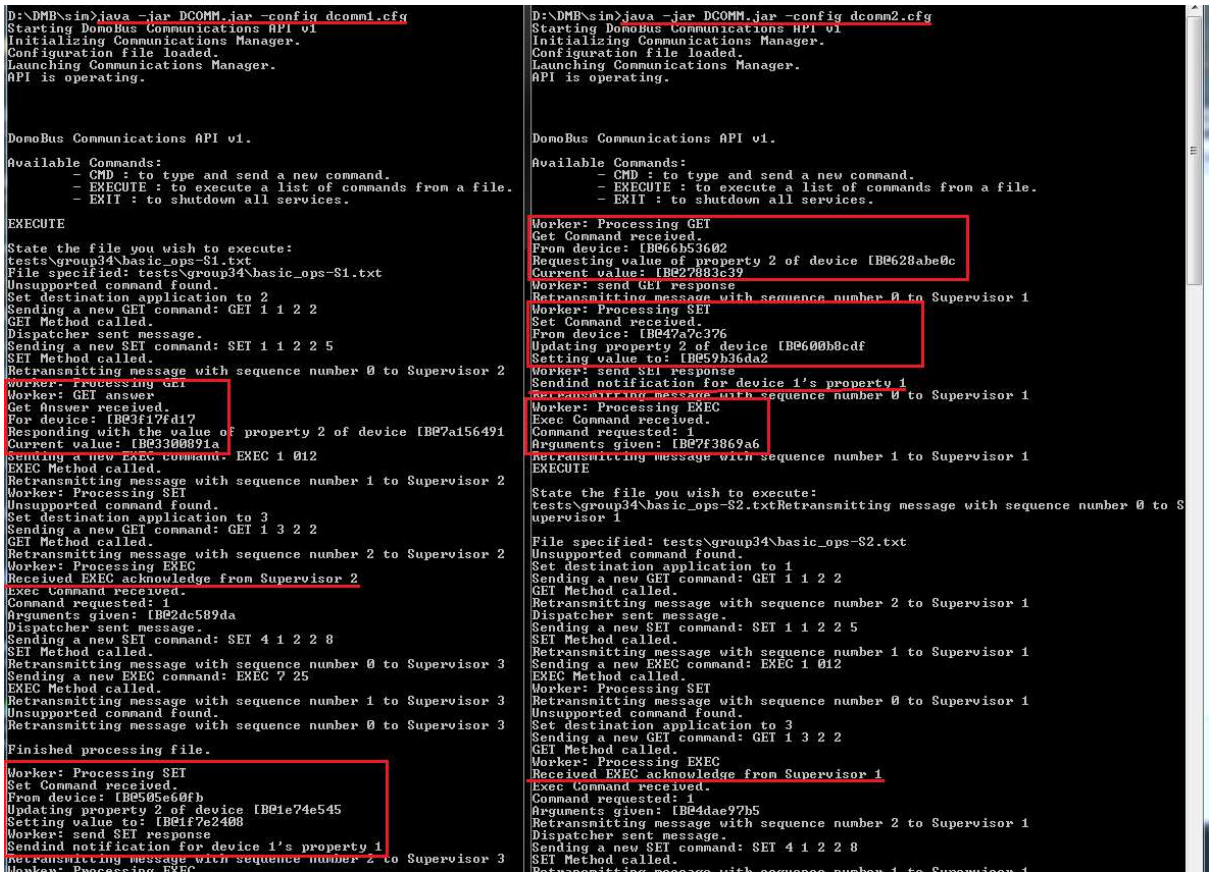


Figure 31 - Asynchronous Tests: Supervisors 1 (left) and 2 (right)

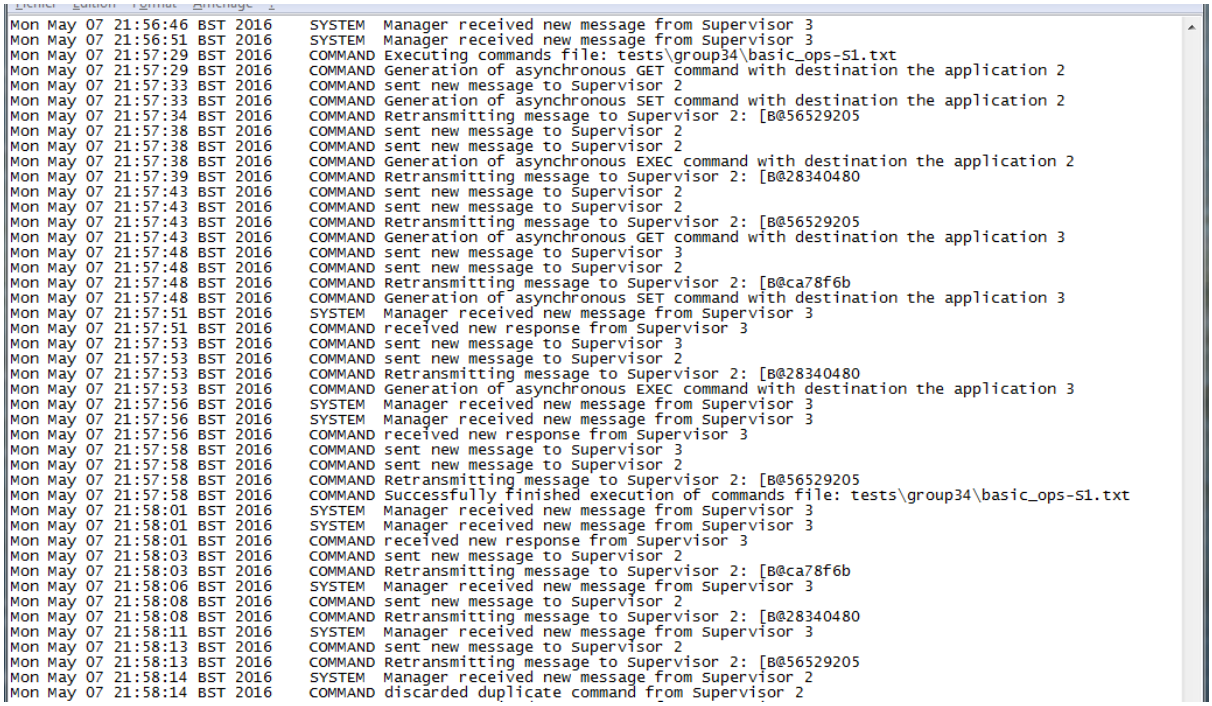


Figure 32 - Asynchronous Tests: Supervisor 1 logs

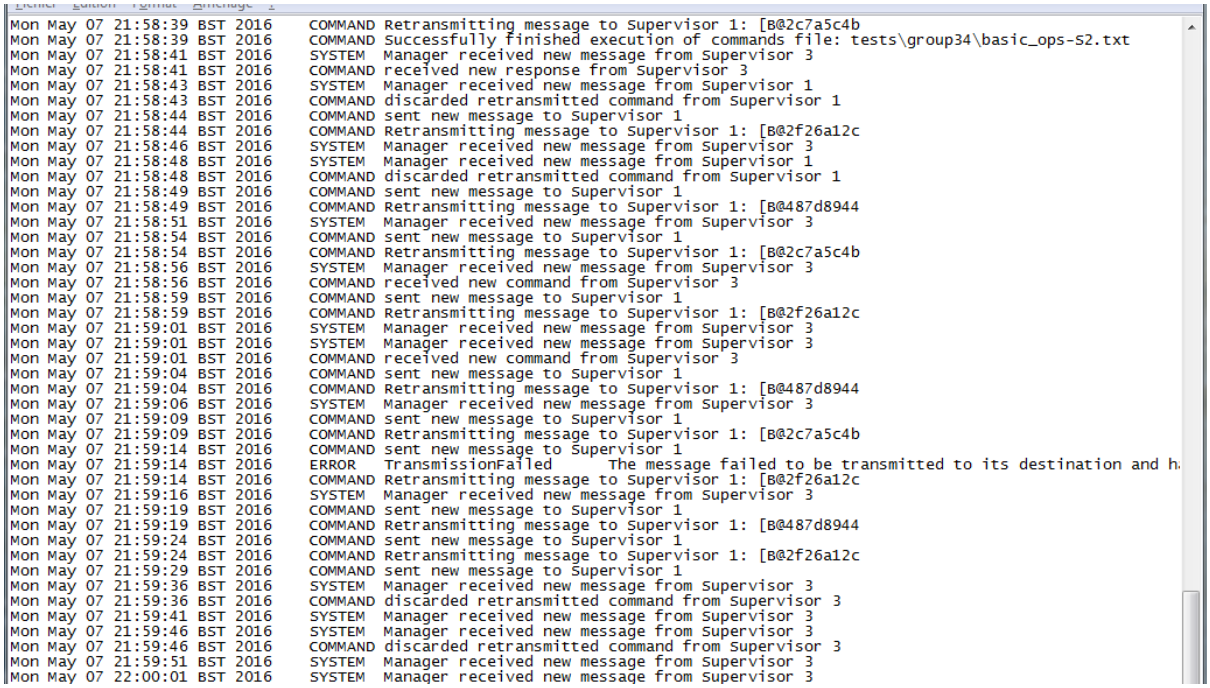


Figure 33 - Asynchronous Tests: Supervisor 2 logs

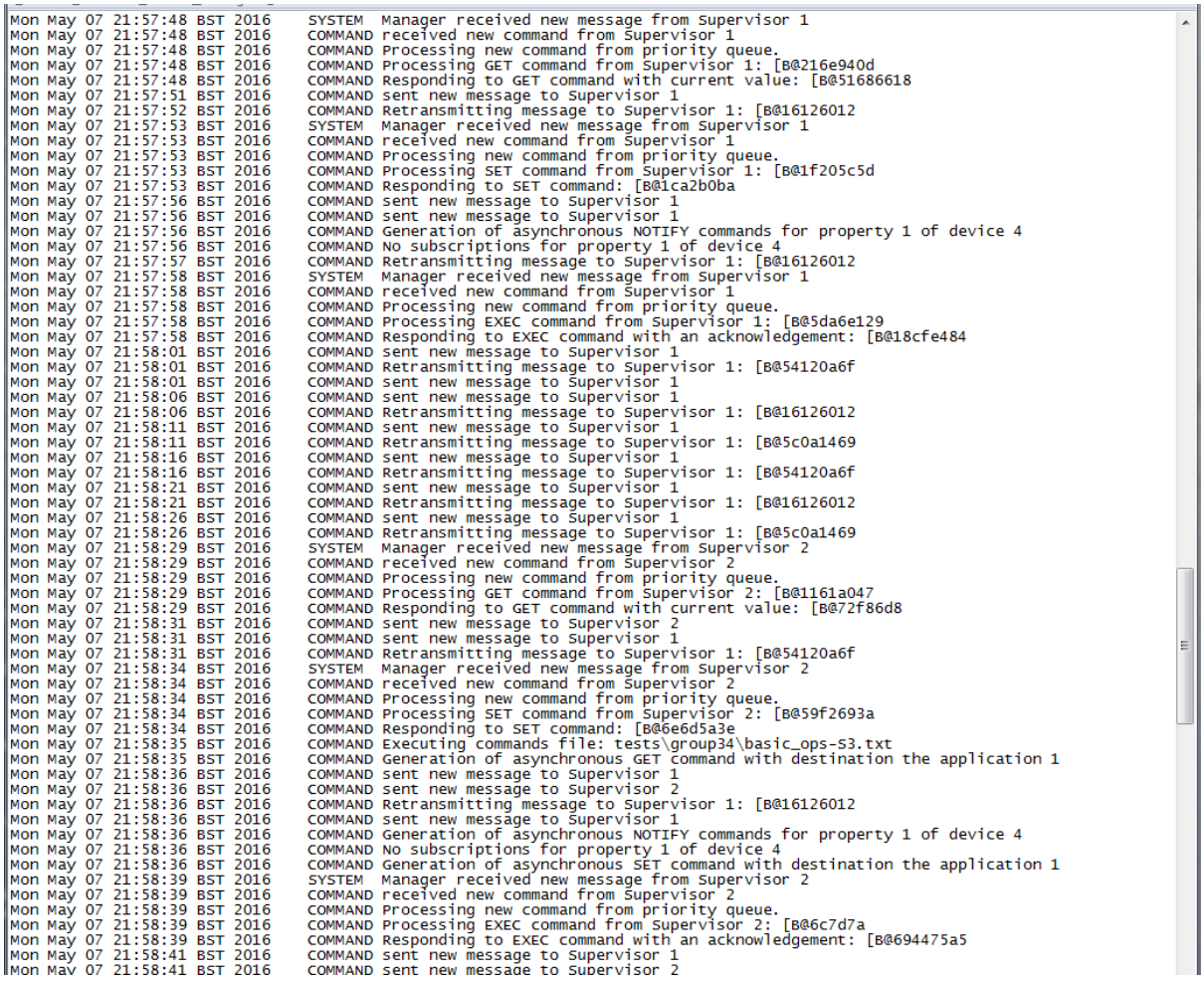


Figure 34 - Asynchronous Tests: Supervisor 3 logs

5.3.4. Synchronous Tests

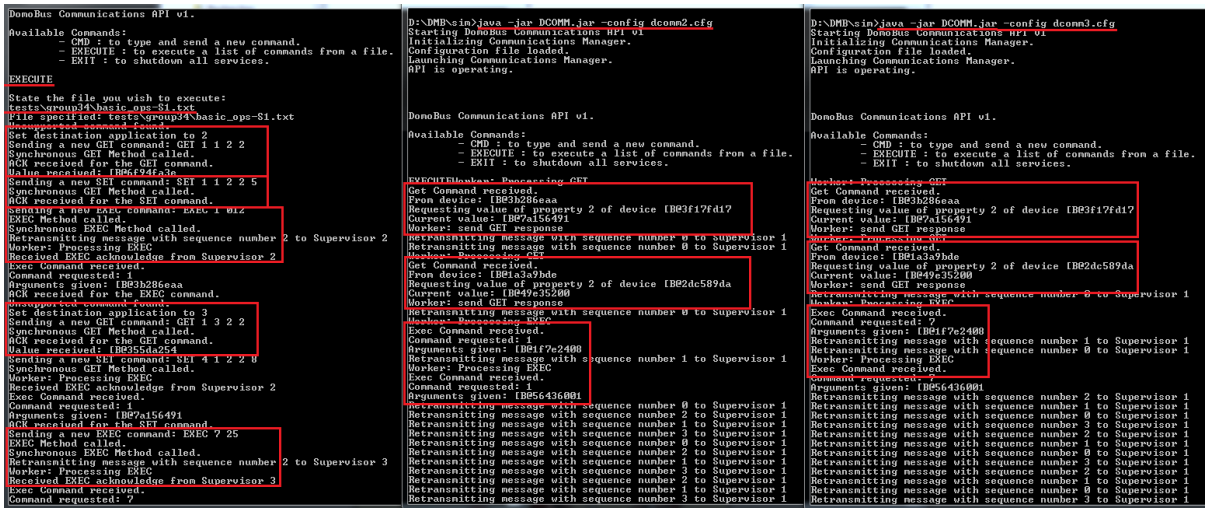


Figure 35 - Synchronous Tests: Supervisors 1 (left), 2 (center) and 3 (right)

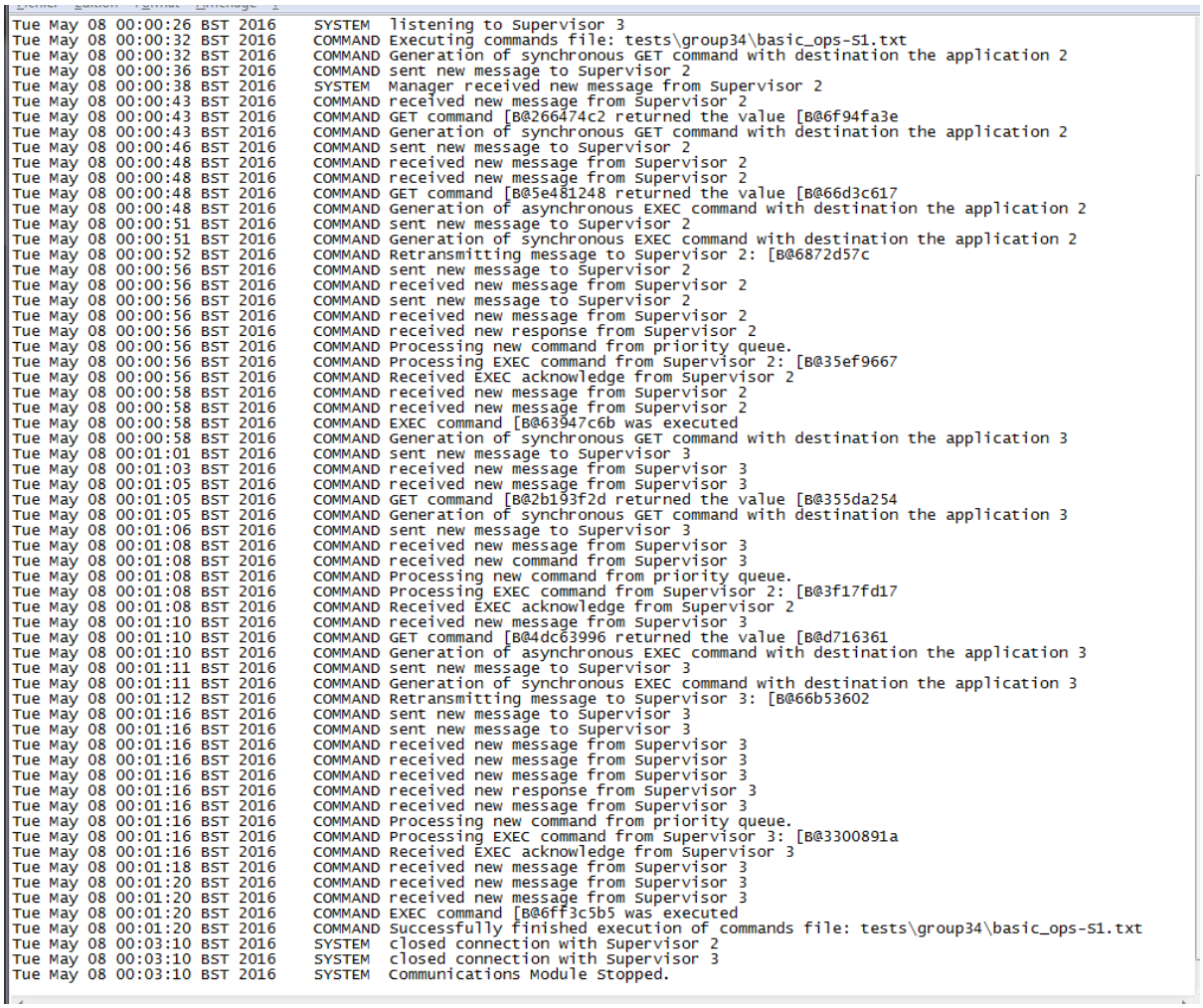


Figure 36 - Synchronous Tests: Supervisor 1 logs

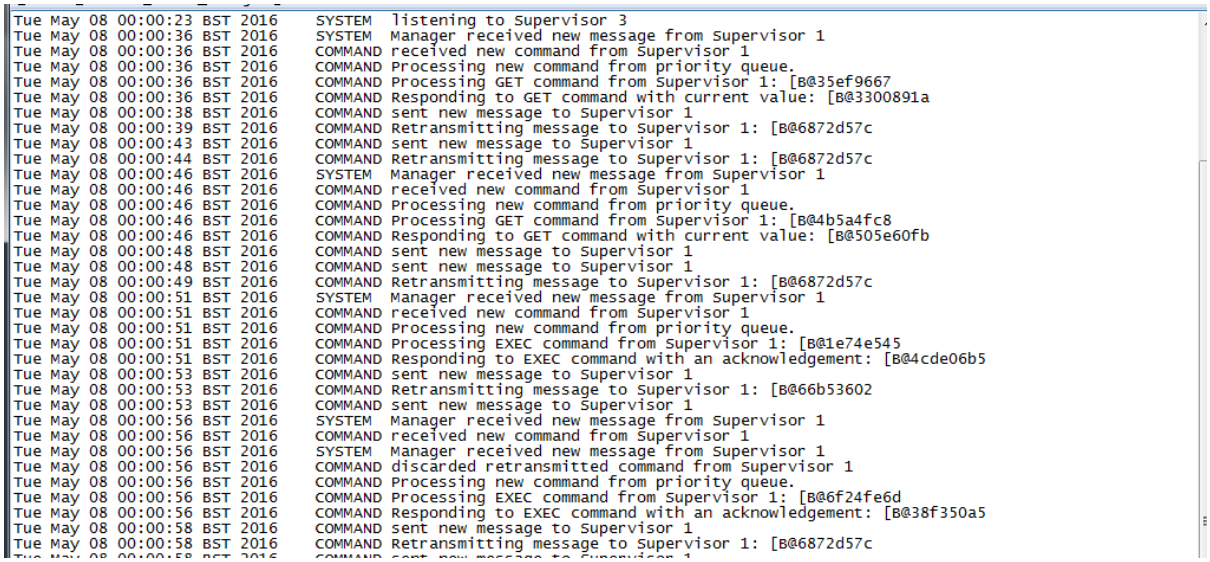


Figure 37 - Synchronous Tests: Supervisor 2 logs

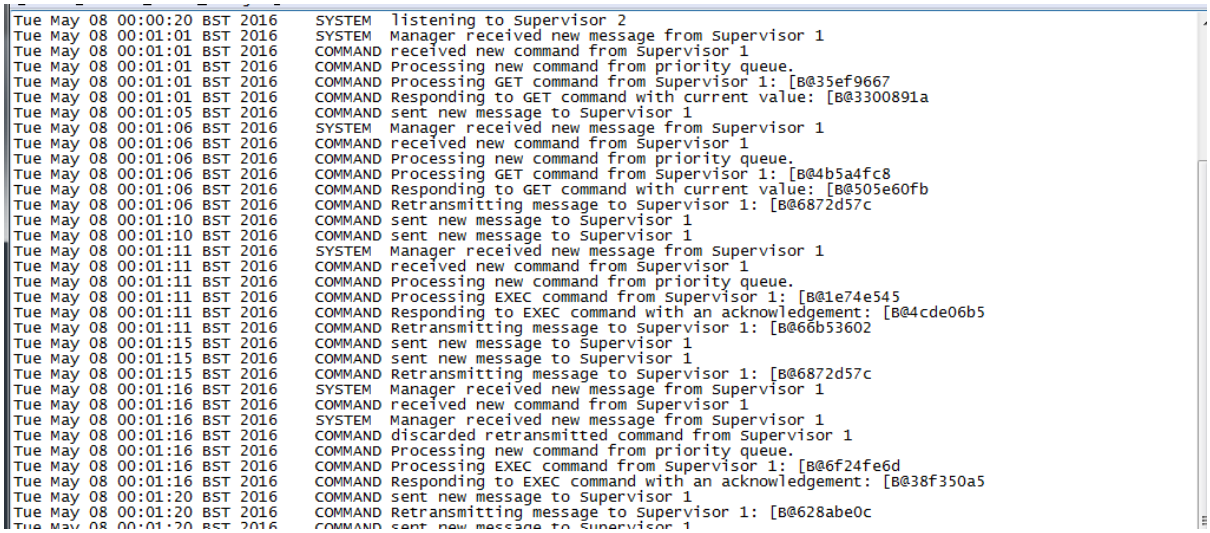


Figure 38 - Synchronous Tests: Supervisor 3 logs

5.3.5. Error Tests

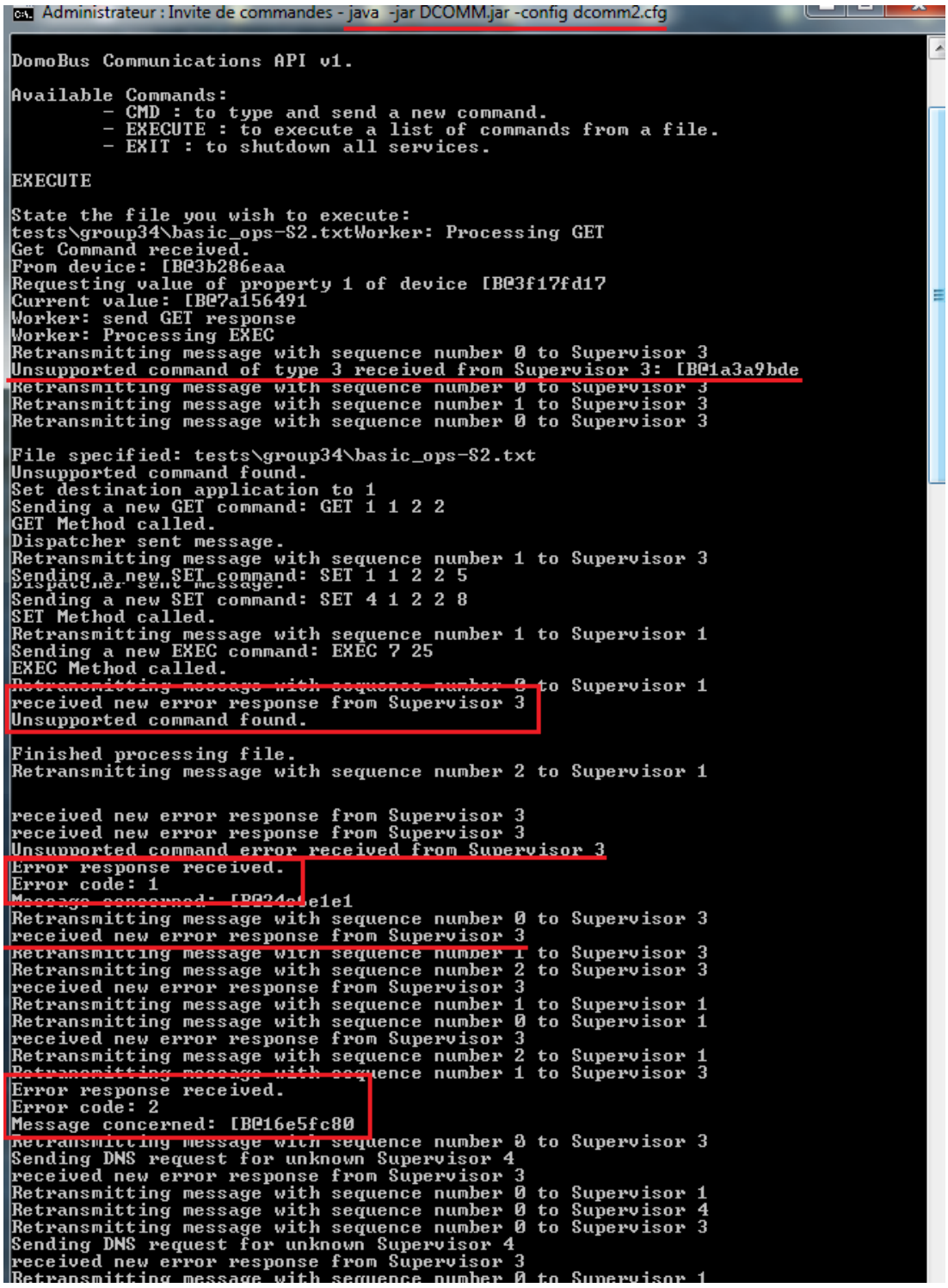


Figure 39 - Error Tests: Supervisor 2

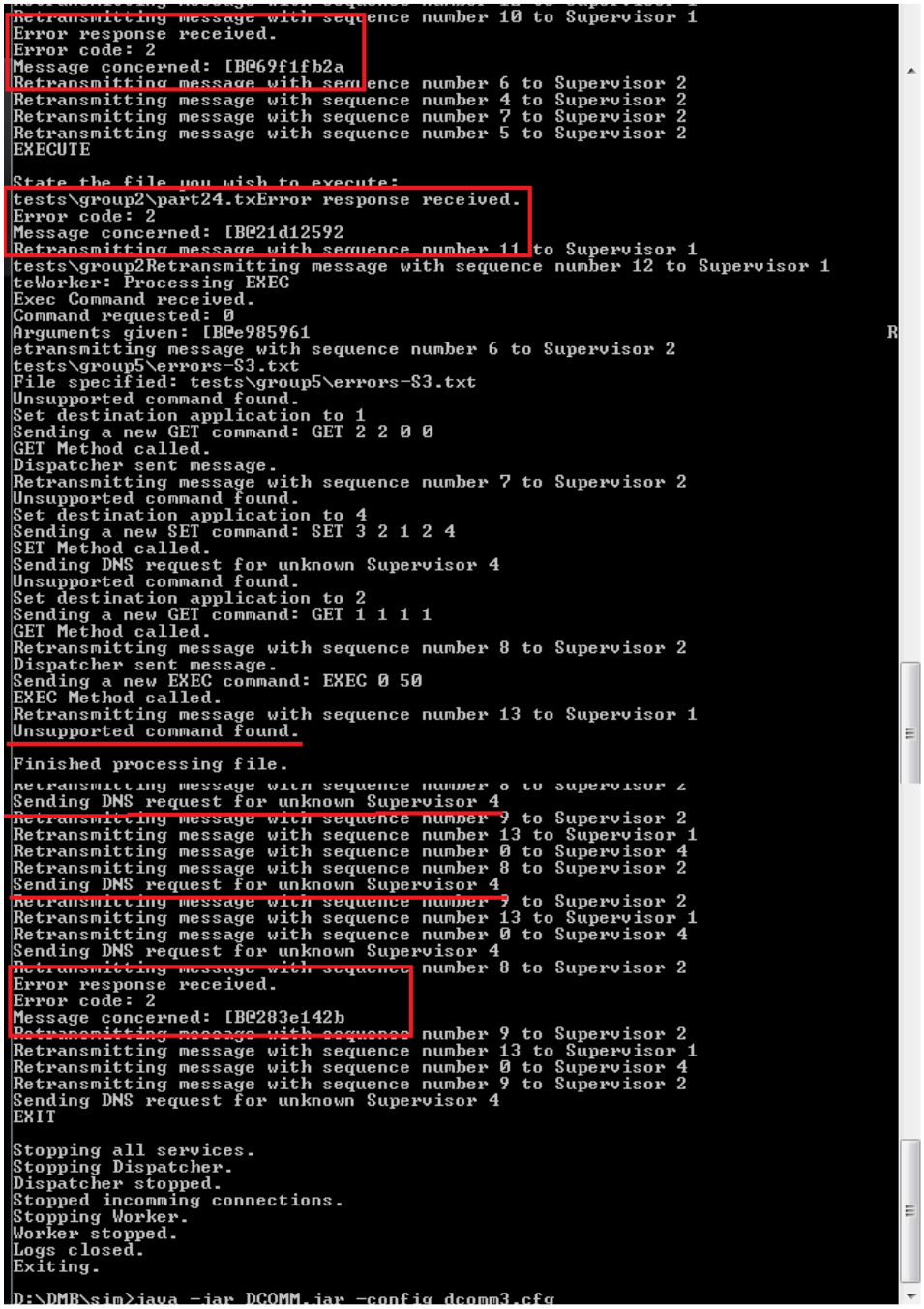
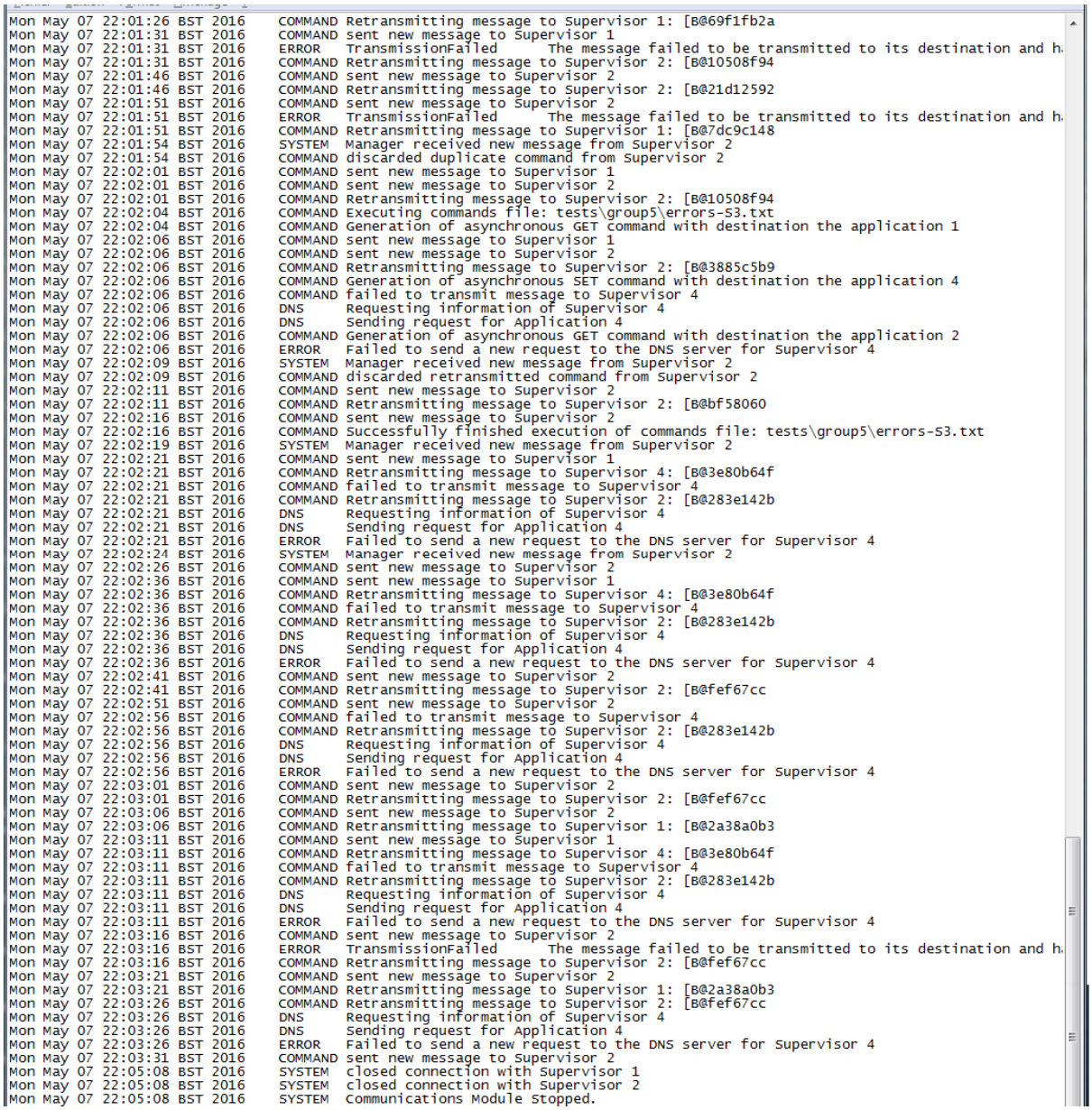


Figure 40 - Error Tests: Supervisor 3

```
Mon May 07 22:01:59 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@56529205
Mon May 07 22:01:59 BST 2016    COMMAND Successfully finished execution of commands file: tests\group5\errors-S2.txt
Mon May 07 22:02:01 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 22:02:01 BST 2016    COMMAND received new response from Supervisor 3
Mon May 07 22:02:04 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 22:02:04 BST 2016    COMMAND Retransmitting message to Supervisor 4: [B@28340480
Mon May 07 22:02:04 BST 2016    COMMAND failed to transmit message to Supervisor 4
Mon May 07 22:02:04 BST 2016    COMMAND Retransmitting message to Supervisor 3: [B@ca78f6b
Mon May 07 22:02:04 BST 2016    DNS     Requesting information of Supervisor 4
Mon May 07 22:02:04 BST 2016    DNS     Sending request for Application 4
Mon May 07 22:02:04 BST 2016    ERROR   Failed to send a new request to the DNS server for Supervisor 4
Mon May 07 22:02:06 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 22:02:06 BST 2016    COMMAND discarded retransmitted command from Supervisor 3
Mon May 07 22:02:09 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 22:02:09 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@56529205
Mon May 07 22:02:09 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 22:02:09 BST 2016    COMMAND Retransmitting message to Supervisor 4: [B@28340480
Mon May 07 22:02:09 BST 2016    COMMAND failed to transmit message to Supervisor 4
Mon May 07 22:02:09 BST 2016    DNS     Requesting information of Supervisor 4
Mon May 07 22:02:09 BST 2016    DNS     Sending request for Application 4
Mon May 07 22:02:09 BST 2016    ERROR   Failed to send a new request to the DNS server for Supervisor 4
Mon May 07 22:02:10 BST 2016    COMMAND Retransmitting message to Supervisor 3: [B@ca78f6b
Mon May 07 22:02:11 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 22:02:11 BST 2016    COMMAND discarded retransmitted command from Supervisor 3
Mon May 07 22:02:11 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 22:02:14 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 22:02:14 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@56529205
Mon May 07 22:02:14 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 22:02:14 BST 2016    COMMAND Retransmitting message to Supervisor 4: [B@28340480
Mon May 07 22:02:14 BST 2016    COMMAND failed to transmit message to Supervisor 4
Mon May 07 22:02:14 BST 2016    DNS     Requesting information of Supervisor 4
Mon May 07 22:02:14 BST 2016    DNS     Sending request for Application 4
Mon May 07 22:02:14 BST 2016    ERROR   Failed to send a new request to the DNS server for Supervisor 4
Mon May 07 22:02:15 BST 2016    COMMAND Retransmitting message to Supervisor 3: [B@ca78f6b
Mon May 07 22:02:16 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 22:02:16 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 22:02:16 BST 2016    COMMAND received new command from Supervisor 3
Mon May 07 22:02:19 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 22:02:19 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@56529205
Mon May 07 22:02:19 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 22:02:19 BST 2016    COMMAND Retransmitting message to Supervisor 4: [B@28340480
Mon May 07 22:02:19 BST 2016    COMMAND failed to transmit message to Supervisor 4
Mon May 07 22:02:19 BST 2016    DNS     Requesting information of Supervisor 4
Mon May 07 22:02:19 BST 2016    DNS     Sending request for Application 4
Mon May 07 22:02:19 BST 2016    ERROR   Failed to send a new request to the DNS server for Supervisor 4
Mon May 07 22:02:20 BST 2016    COMMAND Retransmitting message to Supervisor 3: [B@ca78f6b
Mon May 07 22:02:24 BST 2016    COMMAND sent new message to Supervisor 3
Mon May 07 22:02:24 BST 2016    ERROR   TransmissionFailed      The message failed to be transmitted to its destination and h
Mon May 07 22:02:26 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 22:02:31 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 22:02:31 BST 2016    COMMAND discarded retransmitted command from Supervisor 3
Mon May 07 22:02:41 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 22:02:51 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 22:02:51 BST 2016    COMMAND discarded retransmitted command from Supervisor 3
Mon May 07 22:03:01 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 22:03:06 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 22:03:06 BST 2016    COMMAND discarded retransmitted command from Supervisor 3
Mon May 07 22:03:16 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 22:03:21 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 22:03:21 BST 2016    COMMAND discarded retransmitted command from Supervisor 3
Mon May 07 22:03:31 BST 2016    SYSTEM  Manager received new message from Supervisor 3
Mon May 07 22:03:31 BST 2016    COMMAND discarded retransmitted command from Supervisor 3
Mon May 07 22:05:10 BST 2016    SYSTEM  closed connection with Supervisor 1
Mon May 07 22:05:10 BST 2016    SYSTEM  closed connection with Supervisor 3
Mon May 07 22:05:10 BST 2016    SYSTEM  Communications Module Stopped.
```

Figure 41 - Error Tests: Supervisor 2 logs



```
Mon May 07 22:01:26 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@69f1fb2a
Mon May 07 22:01:31 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 22:01:31 BST 2016    ERROR   TransmissionFailed      The message failed to be transmitted to its destination and h
Mon May 07 22:01:31 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@10508f94
Mon May 07 22:01:46 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 22:01:46 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@21d12592
Mon May 07 22:01:51 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 22:01:51 BST 2016    ERROR   TransmissionFailed      The message failed to be transmitted to its destination and h
Mon May 07 22:01:51 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@7dc9c148
Mon May 07 22:01:54 BST 2016    SYSTEM  Manager received new message from Supervisor 2
Mon May 07 22:01:54 BST 2016    COMMAND discarded duplicate command from Supervisor 2
Mon May 07 22:02:01 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 22:02:01 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 22:02:01 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@10508f94
Mon May 07 22:02:04 BST 2016    COMMAND Executing commands file: tests\group5\errors-S3.txt
Mon May 07 22:02:04 BST 2016    COMMAND Generation of asynchronous GET command with destination the application 1
Mon May 07 22:02:06 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 22:02:06 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 22:02:06 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@3885c5b9
Mon May 07 22:02:06 BST 2016    COMMAND Generation of asynchronous SET command with destination the application 4
Mon May 07 22:02:06 BST 2016    COMMAND failed to transmit message to Supervisor 4
Mon May 07 22:02:06 BST 2016    DNS     Requesting information of Supervisor 4
Mon May 07 22:02:06 BST 2016    DNS     Sending request for Application 4
Mon May 07 22:02:06 BST 2016    COMMAND Generation of asynchronous GET command with destination the application 2
Mon May 07 22:02:06 BST 2016    ERROR   Failed to send a new request to the DNS server for Supervisor 4
Mon May 07 22:02:09 BST 2016    SYSTEM  Manager received new message from Supervisor 2
Mon May 07 22:02:09 BST 2016    COMMAND discarded retransmitted command from Supervisor 2
Mon May 07 22:02:11 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 22:02:11 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@bf58060
Mon May 07 22:02:16 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 22:02:16 BST 2016    COMMAND Successfully finished execution of commands file: tests\group5\errors-S3.txt
Mon May 07 22:02:19 BST 2016    SYSTEM  Manager received new message from Supervisor 2
Mon May 07 22:02:21 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 22:02:21 BST 2016    COMMAND Retransmitting message to Supervisor 4: [B@3e80b64f
Mon May 07 22:02:21 BST 2016    COMMAND failed to transmit message to Supervisor 4
Mon May 07 22:02:21 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@283e142b
Mon May 07 22:02:21 BST 2016    DNS     Requesting information of Supervisor 4
Mon May 07 22:02:21 BST 2016    DNS     Sending request for Application 4
Mon May 07 22:02:21 BST 2016    ERROR   Failed to send a new request to the DNS server for Supervisor 4
Mon May 07 22:02:24 BST 2016    SYSTEM  Manager received new message from Supervisor 2
Mon May 07 22:02:26 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 22:02:36 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 22:02:36 BST 2016    COMMAND Retransmitting message to Supervisor 4: [B@3e80b64f
Mon May 07 22:02:36 BST 2016    COMMAND failed to transmit message to Supervisor 4
Mon May 07 22:02:36 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@283e142b
Mon May 07 22:02:36 BST 2016    DNS     Requesting information of Supervisor 4
Mon May 07 22:02:36 BST 2016    DNS     Sending request for Application 4
Mon May 07 22:02:36 BST 2016    ERROR   Failed to send a new request to the DNS server for Supervisor 4
Mon May 07 22:02:41 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 22:02:41 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@fef67cc
Mon May 07 22:02:51 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 22:02:56 BST 2016    COMMAND failed to transmit message to Supervisor 4
Mon May 07 22:02:56 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@283e142b
Mon May 07 22:02:56 BST 2016    DNS     Requesting information of Supervisor 4
Mon May 07 22:02:56 BST 2016    DNS     Sending request for Application 4
Mon May 07 22:02:56 BST 2016    ERROR   Failed to send a new request to the DNS server for Supervisor 4
Mon May 07 22:03:01 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 22:03:01 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@fef67cc
Mon May 07 22:03:06 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 22:03:06 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@2a38a0b3
Mon May 07 22:03:11 BST 2016    COMMAND sent new message to Supervisor 1
Mon May 07 22:03:11 BST 2016    COMMAND Retransmitting message to Supervisor 4: [B@3e80b64f
Mon May 07 22:03:11 BST 2016    COMMAND failed to transmit message to Supervisor 4
Mon May 07 22:03:11 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@283e142b
Mon May 07 22:03:11 BST 2016    DNS     Requesting information of Supervisor 4
Mon May 07 22:03:11 BST 2016    DNS     Sending request for Application 4
Mon May 07 22:03:11 BST 2016    ERROR   Failed to send a new request to the DNS server for Supervisor 4
Mon May 07 22:03:16 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 22:03:16 BST 2016    ERROR   TransmissionFailed      The message failed to be transmitted to its destination and h
Mon May 07 22:03:16 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@fef67cc
Mon May 07 22:03:21 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 22:03:21 BST 2016    COMMAND Retransmitting message to Supervisor 1: [B@2a38a0b3
Mon May 07 22:03:26 BST 2016    COMMAND Retransmitting message to Supervisor 2: [B@fef67cc
Mon May 07 22:03:26 BST 2016    DNS     Requesting information of Supervisor 4
Mon May 07 22:03:26 BST 2016    DNS     Sending request for Application 4
Mon May 07 22:03:26 BST 2016    ERROR   Failed to send a new request to the DNS server for Supervisor 4
Mon May 07 22:03:31 BST 2016    COMMAND sent new message to Supervisor 2
Mon May 07 22:05:08 BST 2016    SYSTEM  closed connection with Supervisor 1
Mon May 07 22:05:08 BST 2016    SYSTEM  closed connection with Supervisor 2
Mon May 07 22:05:08 BST 2016    SYSTEM  Communications Module Stopped.
```

Figure 42 - Error Tests: Supervisor 3 logs

5.3.6. Configuration Files

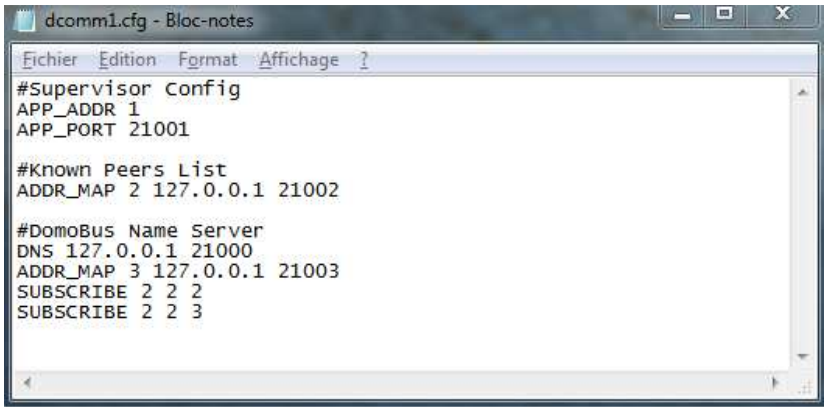


Figure 43 - Results: dcomm1.cfg

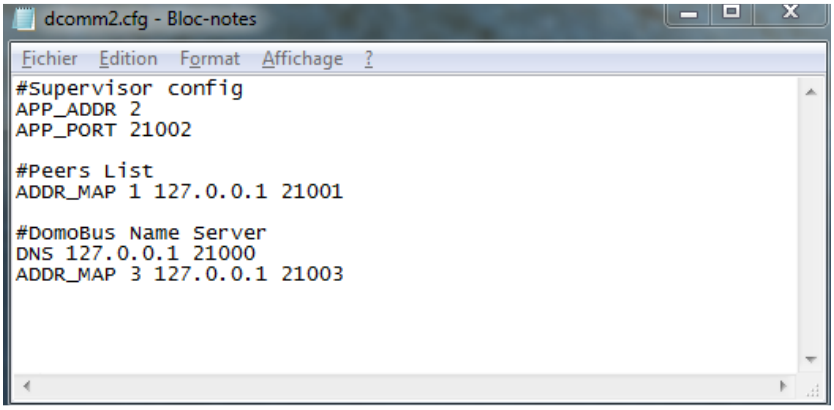


Figure 44 - Results: dcomm2.cfg

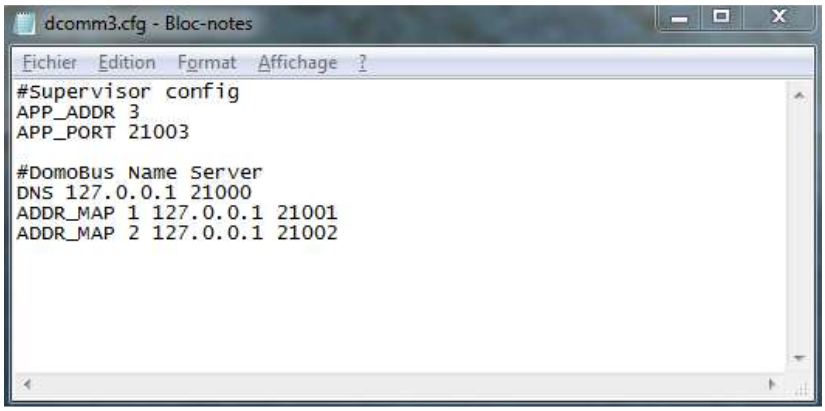


Figure 45 - Results: dcomm3.cfg

5.4. Analysis

The outcome of all those tests were quite positive, as their results were actually able to demonstrate the proper functioning of all the implemented features, whether they were provided by the communications library or the extensions created specifically for the Test Version. We could observe how each Supervisor acted within the network and have a look how they operated internally.

For example, we can see the DNS registrations and queries being sent and the corresponding responses being received, thus, in practice, effectively rending the network capable of dynamic adjustments. And also what happens when a DomoBus Name Server is absent from the network, as shown through the resulting logs of the error tests.

Furthermore, by examining the results, we can attest that both manners of transmitting requested commands, either asynchronously or synchronously, function as desired. With the later cases blocking the sequences of commands, while waiting for the proper response. And that, more generally, all supported operation types are correctly executed and processed.

The subscriptions system works completely as we were able to dynamically manage the subscriptions within each Supervisor through the implemented DCOMM functions, albeit with the exception of the available publishers listing function, which however was deemed as a pure supervision aspect and so outside the scope targeted. Furthermore this dynamism was confirmed with the notification commands when certain NOTIFYs, pertaining to device-property pairs that ceased to have subscribers, stopped being broadcasted. Also, when SET commands were processed, we observed that the subsequent NOTIFY commands where requested afterwards.

The underlying mechanisms and protocols were also noticeable, as we saw packets being retransmitted and even duplicates being discarded. Errors being shown and handled without affecting the Supervisor's functions, substantiating fault-tolerance properties.

Or more plainly the side tasks, like the update of the configuration files, whose contents were extended as shown previously. We can even count the, obviously proven, event-logging system that we were using to monitor the tests results.

As for the test version specifically, the user interface functionalities: individual commands, commands file processing and Supervisor shut down; all operated as they were intended.

5.4.1. Issues

Even though every single underlying aspect and functionality worked correctly on at least one occasion, we must mitigate the results as some direct and indirect issues present.

A minor one was the readability of the consoles output and sometimes even of the logged events, in part due to the other issues discussed below that generated repetitions of similar events. Although it was intended, when using the Test Version, to display detailed information as to provide real-time feedback of the interactions during the simulation. Within the library however, all that feedback is only visible through the logs, while only the services status are shown in the console.

Another issue that is striking concerns the packets transmissions. Specifically the redundancy of retransmissions and it is totally due to the network transport protocol used, UDP, and the retransmissions protocols implemented. Since retransmission is required, all of the acknowledgement packets, and any other response messages, are retransmitted the maximum number of times allowed, flooding the network until they are discarded. Which is a straightforward issue, because there aren't acknowledgements of the acknowledgements. However this problem is in part responsible for the readability issue and could even generate some performance issues in a more busy or populated network, or even as Supervisors could be small computers with limited resources and capabilities.

Finally, we can also note that the concurrency of packets exchanged at practically the same time can cause numbering faults of the shared sequences and eventually lead to non-processed commands.

6. Conclusion

Overall the project's goal was reached, as we were able to build a fully operational DomoBus communications library to act as the communications module of Supervisor applications.
And the simulation created demonstrated the feasibility or at least provided a proof of concept of the Supervision Level network and, more generally, I might say, of the high-level part of the DomoBus network and DomoBus technology.

Realistically, some further optimizations are most probably required, since some mechanisms and functionalities might not have been implemented using the best approach or even the values of the constants used could possibly differ from the optimal conditions during real-world applications.

6.1. Future Work

Relatively to the DomoBus environment as a whole, the scope of this project is quite limited, restricted. This and the fact that the DomoBus technology is still being developed and tested, offers the possibility of extending the work accomplished through this project to multiple different routes.

The more straightforward ones could be implementing and developing the remaining Supervisor modules with the goal of effectively having a finished Supervisor application. Those modules would certainly include an actual graphical user interface and a supervision module. Which through it developing the control-level gateways or even gateways for integrating other home automation technologies would be possible and more easily attainable. A fully complete Supervisor would then allow bridging into the low-level DomoBus technology to establish a full-fledged automation network.

Another way could involve the reinforcement of the network's security as simple flaws are present. For example, DNS registrations could be hijacked by anyone claiming to be a certain application address and alter the corresponding IP address and public port.

Other possibilities could be related to the development and supply of customized services, their management and discovery or even implementations of the remaining operations types currently unused.

7. References

[1] *DomoBus Architecture and Packets formats v2, DomoBus Official webpage*. Retrieved from: http://domobus.net/papers/Architecture_and_Packets_Formats.zip/. Last accessed May 2016.

[2] *Java Platform, Standard Edition 7: API Specification, Oracle Official webpage*. Retrieved from: https://docs.oracle.com/javase/7/docs/api/. Last accessed May 2016.

[3] M. Fourment, M. R. Gillings. A comparison of common programming languages used in bioinformatics. *BMC Bioinformatics*, 2008, pp. 9:82, 5 February 2008.

[4] R. Nunes, *Decentralized Supervision for Home Automation*, MELECON 2006 - The 13th IEEE Mediterranean Electrotechnical Conference, Benalmádena, Spain, May 2006.

8. Annexes

8.1. IDComm Methods Specification

8.1.1. Asynchronous Outgoing Commands

DComm_send_msg_GET	Data Type	Name	Description
Input arguments	Integer	appId	The target application address
	Byte array	DevAddrOrig	Origin device's address
	Byte	PropDescOrig	Origin device property's description
	Byte array	DevAddrDest	Destination device's address
	Byte	PropDescDest	Destination device property's description
	Boolean	Priority	Priority request
Output	Void	N/A	N/A

DComm_send_msg_SET	Data Type	Name	Description
Input arguments	Integer	appId	The target application address
	Byte array	DevAddrOrig	Origin device's address
	Byte	PropDescOrig	Origin device property's description
	Byte array	DevAddrDest	Destination device's address
	Byte	PropDescDest	Destination device property's description
	Byte array	value	New value for the destination property
	Boolean	Priority	Priority request
Output	Void	N/A	N/A

DComm_send_msg_NOTIFY	Data Type	Name	Description
Input arguments	Integer	Dev	Device's address
	Byte	PropDesc	Device property's description
	Byte array	value	New value of the stated property
	Boolean	Priority	Priority request
Output	Void	N/A	N/A

DComm_send_msg_EXEC	Data Type	Name	Description
Input arguments	Integer	appId	The target application address
	Byte	command	The remote function identifier
	Byte array	arguments	The arguments to be used
	Boolean	Priority	Priority request
Output	Void	N/A	N/A

DComm_send_msg_DCOMM	Data Type	Name	Description
Input arguments	Integer	appId	The target application address
	Byte	command	The remote DCOMM function identifier
	Byte array	arguments	The arguments to be used
	Boolean	Priority	Priority request
Output	Void	N/A	N/A

DComm_send_msg_RESERVED	Data Type	Name	Description
Input arguments	Integer	appId	The target application address
	Byte	operation	The remote operation identifier
	Byte array	data	The data to be used
	Boolean	Priority	Priority request
Output	Void	N/A	N/A

8.1.2. Synchronous Outgoing Commands

DComm_send_sync_msg_GET	Data Type	Name	Description
Input arguments	Integer	appId	The target application address
	Byte array	DevAddrOrig	Origin device's address
	Byte	PropDescOrig	Origin device property's description
	Byte array	DevAddrDest	Destination device's address
	Byte	PropDescDest	Destination device property's description
	Boolean	Priority	Priority request
Output	Object array	Byte	Acknowledgement or Error CTR
		Byte array	Response's value or error code

DComm_send_sync_msg_SET	Data Type	Name	Description
Input arguments	Integer	appId	The target application address
	Byte array	DevAddrOrig	Origin device's address
	Byte	PropDescOrig	Origin device property's description
	Byte array	DevAddrDest	Destination device's address
	Byte	PropDescDest	Destination device property's description
	Byte array	value	New value for the destination property
	Boolean	Priority	Priority request
Output	Object array 2 elements	Byte	Acknowledgement or Error CTR
		Byte array	Response's value or error code

DComm_send_sync_msg_NOTIFY	Data Type	Name	Description
Input arguments	Integer	Dev	Device's address
	Byte	PropDesc	Device property's description
	Byte array	value	New value of the stated property
	Boolean	Priority	Priority request
Output	Byte	N/A	Acknowledgement or Error CTR

DComm_send_sync_msg_EXEC	Data Type	Name	Description
Input arguments	Integer	appId	The target application address
	Byte	command	The remote function identifier
	Byte array	arguments	The arguments to be used
	Boolean	Priority	Priority request
Output	Object array 2 elements	Byte	Acknowledgement or Error CTR
		Byte array	Response's value or error code

DComm_send_sync_msg_DCOMM	Data Type	Name	Description
Input arguments	Integer	appId	The target application address
	Byte	command	The remote DCOMM function identifier
	Byte array	arguments	The arguments to be used
	Boolean	Priority	Priority request
Output	Object array 2 elements	Byte	Acknowledgement or Error CTR
		Byte array	Response's value or error code

DComm_send_sync_msg_RESERVED	Data Type	Name	Description
Input arguments	Integer	appId	The target application address
	Byte	operation	The remote operation identifier
	Byte array	data	The data to be used
	Boolean	Priority	Priority request
Output	Object array 2 elements	Byte	Acknowledgement or Error CTR
		Byte array	Response's value or error code

8.1.3. Incoming Commands

DComm_process_callback_GET	Data Type	Name	Description
Input arguments	Byte array	DevAddrOrig	Origin device's address
	Byte	PropDescOrig	Origin device property's description
	Byte array	DevAddrDest	Destination device's address
	Byte	PropDescDest	Destination device property's description
Output	Byte array	N/A	Property's value

DComm_process_callback_ANSWER_GET	Data Type	Name	Description
Input arguments	Byte array	DevAddrOrig	Origin device's address
	Byte	PropDescOrig	Origin device property's description
	Byte array	DevAddrDest	Destination device's address
	Byte	PropDescDest	Destination device property's description
	Byte array	value	Current property's value
Output	Byte array	N/A	Property's value

DComm_process_callback_SET	Data Type	Name	Description
Input arguments	Byte array	DevAddrOrig	Origin device's address
	Byte	PropDescOrig	Origin device property's description
	Byte array	DevAddrDest	Destination device's address
	Byte	PropDescDest	Destination device property's description
	Byte array	value	New value for the destination property
Output	Byte	N/A	Acknowledgement or Error CTR

DComm_process_callback_NOTIFY	Data Type	Name	Description
Input arguments	Byte array	DevAddr	Device's address
	Byte	PropDesc	Device property's description
	Byte array	value	New value of the stated property
Output	Void	N/A	N/A

DComm_process_callback_EXEC	Data Type	Name	Description
Input arguments	Byte	command	The remote function identifier
	Byte array	arguments	The arguments to be used
Output	Boolean	N/A	Successful execution indicator

DComm_process_callback_RESERVED	Data Type	Name	Description
Input arguments	Byte	operation	The remote operation identifier
	Byte array	data	The data to be used
Output	Boolean	N/A	Successful execution indicator

DComm_process_callback_ERROR	Data Type	Name	Description
Input arguments	Integer	errorCode	Code of the occurred error
	Byte array	message	Error details
Output	Void	N/A	N/A

8.2. DCOMM Supervision Level Packet Formats – DNS

8.2.1. Registration Packets

Request Command: Operation Code: 111 (DCOMM) – CTR: 10000111

1 Byte	2 Bytes	2 Bytes	1 Byte	1 Byte	2 Bytes	1 Byte
TLen	null	AppOrig	SNum	CTR	Port	CRC

Remarks: the data field is only composed of the Supervisor's public port number, since its IP address is extractable directly from the socket the DNS has. The priority bit is irrelevant has each new request the DNS receives should be processed directly and the 2 byte pertaining to the destination's application address should be null.

Response Command: Operation Code: 111 (DCOMM) – CTR: 100x1111

1 Byte	2 Bytes	2 Bytes	1 Byte	1 Byte	1 Byte
TLen	AppDest	null	SNum	CTR	CRC

Remarks: the actual DNS response is contained within the control field, as the value of its error bit is enough to indicate if the request was successfully processed or an error occurred. This time the origin address field should be set to null.

8.2.2. Query Packets

Request Command: Operation Code: 111 (DCOMM) – CTR: 10000111

1 Byte	2 Bytes	2 Bytes	1 Byte	1 Byte	2 Bytes	1 Byte
TLen	null	AppOrig	SNum	CTR	AppId	CRC

Remarks: the data field, identified as "AppId", is the Supervision Level address of the wanted Supervisor.

Positive Response Command: Operation Code: 111 (DCOMM) – CTR: 10001111

1 Byte	2 Bytes	2 Bytes	1 Byte	1 Byte	4 Bytes	2 Bytes	1 Byte
TLen	AppDest	null	SNum	CTR	IP	Port	CRC

Remarks: If the requested Supervisor was found, the DNS responds with its IPv4, the dots are excluded and each group of numbers (0-255) are encoded in an individual byte, and its public port.

Negative Response Command: Operation Code: 111 (DCOMM) – CTR: 10011111

1 Byte	2 Bytes	2 Bytes	1 Byte	1 Byte	2 Bytes	1 Byte
TLen	AppDest	null	SNum	CTR	AppId	CRC

Remarks: if the requested Supervisor was not found or the address given is incorrect, the DNS sends back the requested address and toggles both the acknowledgement and error bits of the CTR.

8.3. DCOMM Supervision Level Packet Formats – Subscriptions

8.3.1. Subscribe

Request Command: Operation Code: 111 – CTR: 1x000111 – Function: **DCOMM_SUBSCRIBE**

1 Byte	2 Bytes	2 Bytes	1 Byte	1 Byte	1 Byte	1 Byte	2 Bytes	1 Byte	1 Byte
TLen	AppDest	AppOrig	SNum	CTR	Length	Function	DevAddr	PropDesc	CRC

8.3.2. Unsubscribe

Request Command: Operation Code: 111 – CTR: 1x000111 – Function: **DCOMM_UNSUBSCRIBE**

1 Byte	2 Bytes	2 Bytes	1 Byte	1 Byte	1 Byte	1 Byte	2 Bytes	2 Bytes	1 Byte
TLen	AppDest	AppOrig	SNum	CTR	Length	Function	DevAddr	PropDesc	CRC

8.3.3. Unsubscribe All

Request Command: Operation Code: 111 – CTR: 1x000111 – Function: **DCOMM_UNSUBSCRIBE_ALL**

1 Byte	2 Bytes	2 Bytes	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte
TLen	AppDest	AppOrig	SNum	CTR	Length	Function	CRC

Remarks: unsubscribes from any existing subscriptions at the destination's Supervisor.

8.3.4. List Publishers

Request Command: Operation Code: 111 – CTR: 1x000111 – Function: **DCOMM_LIST_PUBLISHERS**

1 Byte	2 Bytes	2 Bytes	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte
TLen	AppDest	AppOrig	SNum	CTR	Length	Function	CRC

Remarks: requests all the existing device-property pairs that can be subscribed at the specified application.

8.4. Global Variables and Constants

8.4.1. Category: Configuration File

Refer to Chapter 4, 4.3.1.

8.4.2. Category: Supervision Level Packets

Operation Codes: **Refer to Chapter 3, 3.2.3.**

Name	Value	Description
CTR	0x80	Base control field byte.
PRIORITY_CTR	0x40	Control field priority bit.
RETRANSMISSION_CTR	0x20	Control field retransmission bit.
ERROR_CTR	0x10	Control field error bit.
ACK_CTR	0x08	Control field acknowledgement bit.
PRIORITY_POS	6	Position of the priority bit in the control field, 0 being the least significant bit.
RETRANSMISSION_POS	5	Position of the retransmission bit in the control field, 0 being the least significant bit.
ERROR_POS	4	Position of the error bit in the control field, 0 being the least significant bit.
ACK_POS	3	Position of the acknowledgement bit in the control field, 0 being the least significant bit.
MASK_8BIT	0x00	8-Bits property type bits.
MASK_16BIT	0x40	16-Bits property type bits.
MASK_ARRAY	0x80	DomoBus array property type bits.
MASK_RESERVED	0xC0	Reserved property type bits.
INVALID_VALUE	0x20	Invalid value property type bit.
MIN_PACKET_LENGTH	8	Shortest Supervision Level Packet length.
MAX_PACKET_LENGTH	255	Longest Supervision Level Packet length.
DNS_REGISTER	0x00	DNS registration function code.
DNS_GET	0x01	DNS request function code.
DCOMM_SUBSCRIBE	0x02	DCOMM subscription function code.
DCOMM_UNSUBSCRIBE	0x03	DCOMM unsubscribe function code.
DCOMM_UNSUBSCRIBE_ALL	0x04	DCOMM unsubscribe from all function code.
DCOMM_LIST_PUBLISHERS	0x05	DCOMM list publishers function code.

8.4.3. Category: Errors

Name	Value	Description
ERROR_QUEUES_FULL	0x00	All queues are currently full error code.
ERROR_UNSUPPORTED_COMMAND	0x01	Command requested is not supported error code.
ERROR_TRANSMISSION_FAILED	0x02	All packet transmissions failed error code.
ERROR_TIMEOUT	0x03	Request timed out error code.
ERROR_NOT_FOUND	0x04	Requested data was not found error code.
ERROR_GET_COMMAND	0x10	Generic GET command error code.
ERROR_SET_COMMAND	0x20	Generic SET command error code.
ERROR_NOTIFY_COMMAND	0x30	Generic NOTIFY command error code.
ERROR_EXEC_COMMAND	0x40	Generic EXEC command error code.
ERROR_RESERVED_COMMAND	0x50	Generic RESERVED commands error code.
ERROR_DCOMM_COMMAND	0x80	Generic DCOMM command error code.

8.4.4. Category: Logs

Name	Value	Description
LOG_FOLDER	logs	Name of the folder where the log files are stored.
LOG_EXTENSION	.log	Extension used by the log files.
LOG_PERIOD	86400	Duration of a log file in seconds, equivalent to 1 day.

8.4.5. Category: Network

Name	Value	Description
PRIORITY_QUEUE_SIZE	25	Limit of concurrent messages in the priority queue.
QUEUE_SIZE	100	Limit of concurrent messages in the normal queue.
MAX_RETRANSMISSIONS	5	Number of possible retransmissions before a packet is discarded.
RESTRANSMISSION_PERIOD	500	Waiting interval in milliseconds before another transmission of a given packet is made.
CONNECTION_TIMEOUT	5000	Maximum waiting interval, in milliseconds, before an inactive connection times out.
DNS_REQUEST_TIMEOUT	5000	Maximum waiting interval, in milliseconds, for a DNS response.

8.4.6. Miscellaneous

Name	Value	Description
BACKUP_PERIOD	1800000	Waiting period, in milliseconds, before the configuration file is updated.
VACATION_DURATION	100	Idle interval, in milliseconds, for when queues are empty.

8.5. Simulation: Configuration Files

8.5.1. Supervisor Application 1 – dcomm1.cfg

```
#Supervisor Config
APP_ADDR 1
APP_PORT 21001

#Known Peers List
ADDR_MAP 2 127.0.0.1 21002

#DomoBus Name Server
DNS 127.0.0.1 21000
```

8.5.2. Supervisor Application 2 – dcomm2.cfg

```
#Supervisor Config
APP_ADDR 2
APP_PORT 21002
```

```
#Known Peers List
ADDR_MAP 1 127.0.0.1 21001

#DomoBus Name Server
DNS 127.0.0.1 21000
```

8.5.3. Supervisor Application 3 – dcomm3.cfg

```
#Supervisor Config
APP_ADDR 3
APP_PORT 21003

#DomoBus Name Server
DNS 127.0.0.1 21000
```

LIST OF FIGURES

List Of Tables