TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Daniel El Basania 177266 IASM

# DANFOSS DRIVE SOFTWARE UPGRADER AS A COMMAND LINE TOOL

Master's thesis

Supervisor: Margarita Spitšakova
PhD
Co-Supervisor: Rui Miguel Martins
Costa
MSc

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Daniel El Basania 177266 IASM

# DANFOSSI SAGEDUSMUUNDURI UUENDAJA KÄSUREA TÖÖRIISTANA

Magistritöö

Juhendaja: Margarita Spitšakova
PhD
Kaasjuhendaja: Rui Miguel Martins
Costa
MSc

Tallinn 2019

# Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Daniel El Basania

03.05.2019

# Abstract

This thesis is written in English and is 48 pages long, including four chapters, 22 figures and 12 tables.

Nowadays automation has an important role in terms of technical process management and allows doing more with less effort. The Danfoss Group is a leading manufacturer of energy-efficient automation solutions for enterprises that also includes manufacturing of variable frequency converters (drives).

The master's thesis deals with a considerate extension of features of a software tool called Automation Production Setup Tool (APST). Currently, the tool has the functionality to write binary files and set pre-defined setup parameters values, however, it lacks features that allow upgrading of Danfoss variable-frequency drives firmware.

The purpose of APST is to avoid manual work of dealing with Motion Control Tool (MCT 10) for upgrading the firmware of Danfoss drives. Furthermore, APST ensures error-free usage in the production by using pre-defined setups and absence of user interactions.

# Annotatsioon

## Danfossi sagedusmuunduri uuendaja käsurea tööriistana

Lõputöö on kirjutatud Inglise keeles ning sisaldab teksti 48 leheküljel, 4 peatükki, 22 joonist, 12 tabelit.

# List of abbreviations and terms

| | |
|---|---|
| APST | Automation Production Setup Tool |
| MCT 10 | Motion Control Tool 10 |
| FC | Frequency Converter |
| OEM | Original Equipment Manufacturer |
| ERP | Enterprise Resource Planning |
| COM | Communication port |
| DDComm | Danfoss Drives Communication Module |
| PC | Personal Computer |
| CRC | Cyclic Redundancy Check |
| PLC | Programmable Logic Controller |
| CSV | Comma-Separated Values |
| XML | Extensible Markup Language |
| GUI | Graphical User Interface |
| OSE | Operation Support Engineering |
| USB | Universal Serial Bus |

# Table of contents

# List of figures

# List of tables

# 1 Introduction

Nowadays in the age of automation, it is crucial for worldwide manufacturers of industrial electronics and automatics, like Danfoss, to handle new trends and needs. As production volumes on manufacturing plants grow, new challenges of preserving high standards of quality and reducing production time may become a bottleneck of the production line. The thesis provides one of the solutions on how to increase the quality of the manufacturing department, reduce time overheads and costs.

Danfoss aims to provide energy-efficient solutions for these challenges using variable frequency converters (or frequency drives) designed to control the variable speed of rotation of all types of asynchronous motors and permanent magnet motors [1], therefore, the master's thesis deals with the frequency drives workflow optimization.

To provide the peak production capacity, frequency converters must be properly setup for the tasks to be completed. The setup involves setting the optimal parameters and flashing (upgrading) a frequency converter (FC). The current setup process of frequency converters on an assembly line requires an operator to manually flash a FC to the needed firmware using MCT 10 software tool (Figure 1) that causes time overhead with possible mistakes.

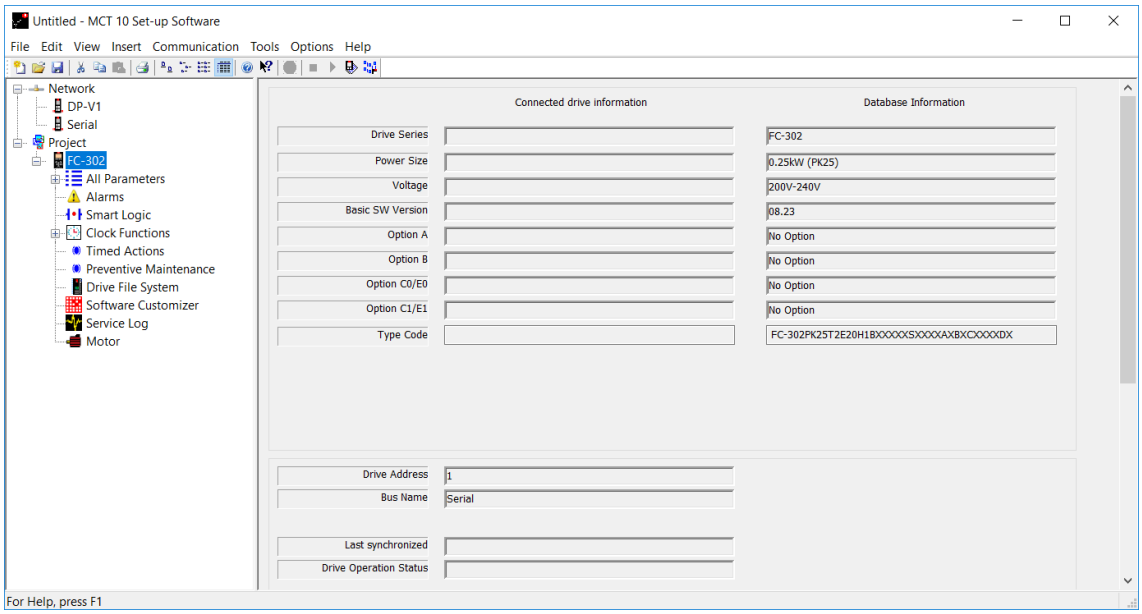



Figure 1. MCT 10 interface.

Aim of the thesis is to extend the functionality of the automation tool to solve the problem of wasting valuable time using MCT 10 and doing extra manual work that eventually may lead to human factor errors.

The thesis is composed of four chapters.

Chapter 2 provides justification of used technologies, description of the APST algorithm used for flashing drives and ends discussing planned features for the future work.

Chapter 3 provides an overview of the tool features, shows how to configure and use APST with the actual use cases.

Chapter 4 summarizes the thesis work.

The thesis contains two appendixes providing additional insight on used date-time formats in APST and an example of a log file using the upgrade feature.

## 1.1 Motivation

MCT 10 has been out on the market for over 15 years now and it enables a full system configuration and control of a frequency converter. With MCT 10 Set-up Software, it is possible to monitor the entire system more effectively for faster diagnosis and better

preventive maintenance and it is designed as an interactive commissioning tool for quick and easy commissioning of various frequency converter series [2]. With its broad feature possibilities, it has become obvious that some features like drive upgrader may not be the best to use on a production line due to time overheads, redundant cautions of cases that never happen on a production line and a probability of making a mistake by an operator. These factors ensue higher costs and lower quality of the end product.

As a solution to the above-described problem, the student working at Proekspert AS [3] was proposed to develop an upgrade feature for APST as a master's thesis project that could automate some of the MCT 10 features, which for the time being are done manually.

The following questions were raised in terms of functional requirements to consider the solution fulfilling the stated problem:
Question I. Is it possible to upgrade a frequency drive with no user interactions or minimal user assistance?
Question II. Is it possible to minimize a possibility of a human-factor error from a factory technician point of view?
Question III. Can APST support both Mark I and Mark II (the first and the second generations) P400 drive family drives?

## 1.2 Technologies used in the development

Throughout the development of APST, the following technologies had been used:

1. C++ 03/11 standards as Danfoss Drives Communication Module (DDComm) is written in C++ and it is a crucial component for the physical communication with drives. The role of the communication module in the connectivity platform is to encapsulate Fieldbus knowledge, i.e. the knowledge about physical buses and protocols that are used to transfer data between a personal computer (PC) and any physical Danfoss drives [4].

2. C++ Qt Framework [5]. The Qt Framework provides an extensive system of classes. The framework was mostly used for a convenient file handling, serializing and deserializing XML files, building the graphical user interface and for date-time formats (see Appendix 1).

3. C++ Boost [6]. Boost represents a set of libraries for the C++ programming language with a wide variety of applications. The library has been used for a more convenient parsing of regular expressions comparing to approaches of the standard library and for cyclic redundancy check (CRC) calculations in the firmware files.

# 2 Automation Production Setup Tool

Automation Production Setup Tool represents an instrument for Danfoss meant to replace MCT 10 with its problems and overheads. The solution is to build an automated firmware upgrader programming tool for the drives, which makes it easier for original equipment manufacturers (OEM) to program the drives on the production floor so that it will reduce production time and increase quality. By increasing quality, it is meant that easy selection (automated via barcode scanning) of a correct file to download to the drive, saves time compared to the look up operation today in MCT 10 and eliminates the risk that the operator is selecting the wrong file to download into the drive.

Planned usage of APST on the production line includes a few steps, specifically, using the MCT 10 tool, an engineer must create project files with specific settings needed for a particular installation. Having done that, MCT 10 compiles it together to a one master file that is linked to the current production system (e.g. enterprise resource planning (ERP) system). The next step is that the engineer on the production line scans the barcode of the drive and that way the correct files are retrieved with the further upgrading of the drive and writing files' content (configurations). At the end of the process, the tool indicates to the engineer whether the operation has been successful [7]. More information about the master file and the tool usage can be found in Chapter 3.

## 2.1 APST algorithm

This subchapter discusses the way APST works in case it is used for flashing, i.e. `-f` argument is used, or a targeted master file contains `FirmwareFile` tag with a valid path to OSE file. The subchapter also provides an insight into the firmware file itself and discusses its handling. Figure 4 shows the simplified version of the APST algorithm as a block scheme diagram. Some details have been omitted due to the signed non-disclosure agreement between the student and Proekspert AS.

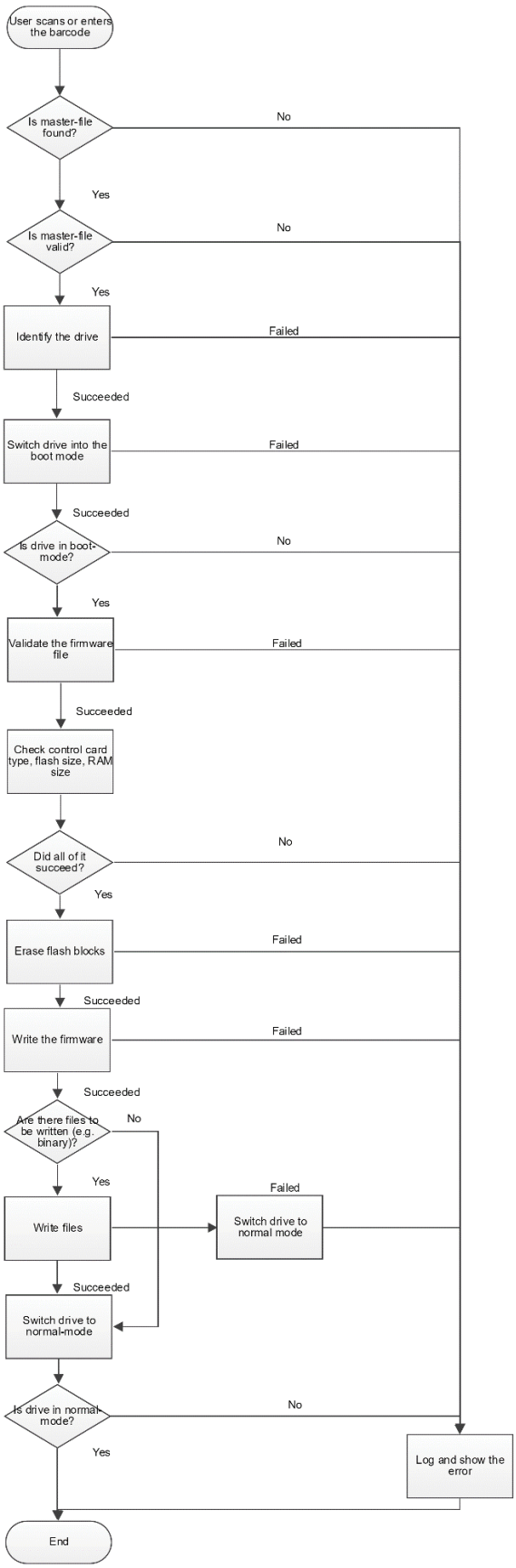



Figure 2. APST algorithm of the flash command.

### 2.1.1 Firmware file

A firmware represents a file of an OSE extension (Operation Support Engineering, named after a Danfoss homonymous department name) that contains scripts to be executed and during processing is extracted in a binary file(s), which are used to flash drives [8].

On an abstract level, every OSE file consists of a number of firmware sections. Each of those sections contains a script and optional firmware data. When using an OSE file all of those sections are used and flashed sequentially. There is no mechanism to choose whether to use a particular section or not [8].

For each section, first, the script is executed and then the firmware data, if any, is sent to the drive. Most drive types, e.g. all of the P400 family, use OSE files with a single section. Drives which require multiple separate firmware files, e.g., FC101, use multiple sections though [8].

The scripts used in OSE files are basically lists of commands sent to the Danfoss drives in the boot mode. They can, however, also contain special meta-commands to allow some simple logic. Those meta-commands include IF statements, commands to display messages to users and to abort the flashing process [8].

In MCT 10, the firmware data is sent to the drive using either Xmodem or Ymodem protocol, though APST always forces Ymodem protocol since Mark II P400 drives do not support Xmodem protocol. If firmware data is present in an OSE section then the script of that section must end with a corresponding “load” command which initiates the data transfer, informing the drive of e.g., the address where the firmware data should be written to.

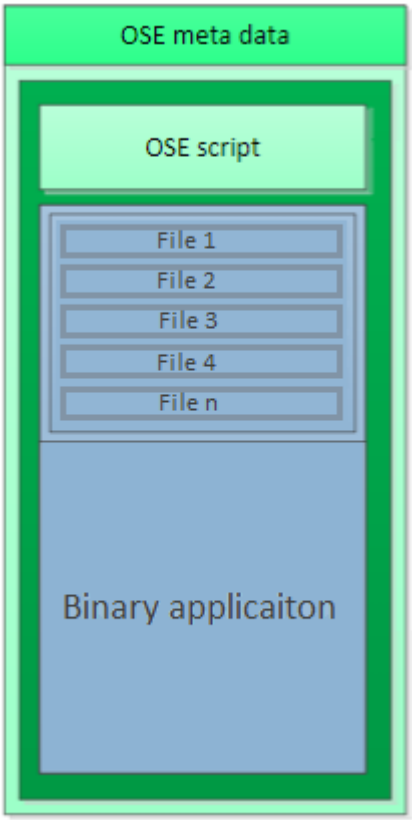



Figure 3. Abstract content of an OSE file.

```
;-------------------------------------------------------
; AOC software update (Advanced CC) OSE command file.
;-------------------------------------------------------

; Check control card type
; Read from OTP address 0x1FFF7800:
; 0: Basic
; 1: Extended
; 0xff: Advanced
; Note: readmem will fail in mkI cards and abort script.
;
dtm readmem -a 0x1fff7800 -b
#### IF!= 0xFF
#### WARN CC Type check - mkII ACC required !
dtm write_ee -a 0 -d 0101 ;added by OssCreator at 2019-03-06T14:29:48
dtm mode_change ;added by OssCreator at 2019-03-06T14:29:48
ABORT
#### END

#### ECHO CC Type check - OK

;-------------------------------------------------------
; Check Flash size and that it is a mkII card
; Note: TM returns "8192 kB" on mkII and "4 MB" on mkI cards
; so it will fail on mkI cards.
;
```

```
dtm flash_size -d 2
#### IF!= 0x2000
#### WARN CC Flash check - 8MB required !
dtm write_ee -a 0 -d 0101 ;added by OssCreator at 2019-03-06T14:29:48
dtm mode_change ;added by OssCreator at 2019-03-06T14:29:48
ABORT
#### END

#### ECHO CC Flash check - OK


;-------------------------------------------------------
; AOC software update (Advanced CC) OSE command file.
;-------------------------------------------------------

#### ECHO CC Flash erase - started...
dtm field_flash_erase -d 2
#### ECHO CC Flash erase - OK

;-------------------------------------------------------
; Flash the drive
load -r -b 0x60020000 -m y
#### ECHO CC Flash update - OK
```

Figure 4. Example of a firmware file content.

### 2.1.2 Scanning a barcode

For drive firmware upgrade using a barcode scanner, a graphical user interface (GUI) must be launched using a `--gui/-g` argument. Before using GUI, a configuration file **config.xml** must be configured and present in the folder with installed APST (see chapter 3 for configuring APST). If APST is configured correctly, the user will see an image very similar to Figure 5.

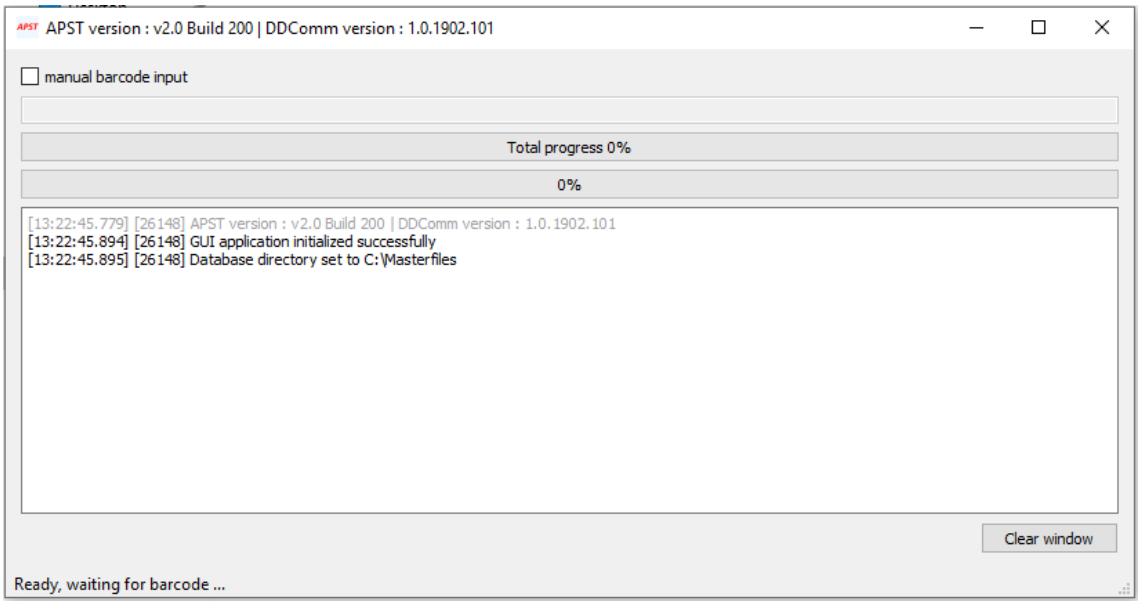



Figure 5. APST GUI interface. The tool is in the ready mode.

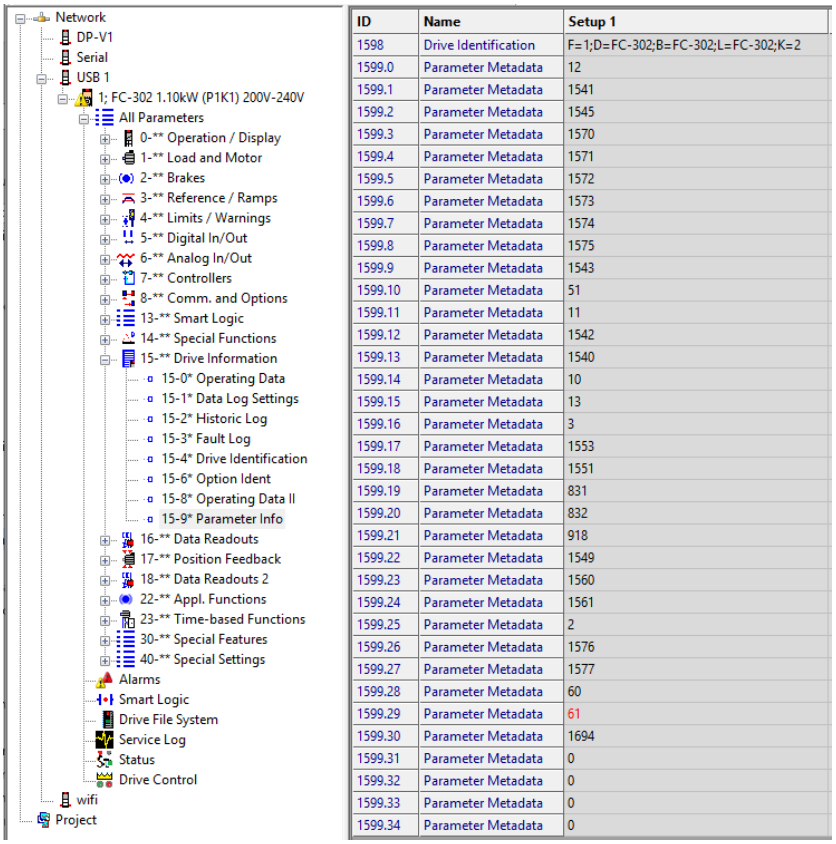
Upon successful launch of APST GUI, the tool is in the ready mode targeting master files in a folder specified in the configuration file config.xml and waiting for a barcode to be scanned. Once the barcode is scanned, APST starts deserialization of the master file with further execution depending on the master file content. Throughout the development process, the "Opticon" barcode scanner had been used.

### 2.1.3 Drive identification

To be able to work with a drive, the drive must be identified and in case of APST, it obliges usage of a serial bus.

The current identification procedure is split into 4 drive family categories: **P400**, **P600**, **MCD5xx** (Soft Starter) and **VLT** [9].

The division is based on the "F" value of the response and obtained by reading parameter 1598.




| ID | Name | Setup 1 |
|---|---|---|
| 1598 | Drive Identification | F=1;D=FC-302;B=FC-302;L=FC-302;K=2 |
| 1599.0 | Parameter Metadata | 12 |
| 1599.1 | Parameter Metadata | 1541 |
| 1599.2 | Parameter Metadata | 1545 |
| 1599.3 | Parameter Metadata | 1570 |
| 1599.4 | Parameter Metadata | 1571 |
| 1599.5 | Parameter Metadata | 1572 |
| 1599.6 | Parameter Metadata | 1573 |
| 1599.7 | Parameter Metadata | 1574 |
| 1599.8 | Parameter Metadata | 1575 |
| 1599.9 | Parameter Metadata | 1543 |
| 1599.10 | Parameter Metadata | 51 |
| 1599.11 | Parameter Metadata | 11 |
| 1599.12 | Parameter Metadata | 1542 |
| 1599.13 | Parameter Metadata | 1540 |
| 1599.14 | Parameter Metadata | 10 |
| 1599.15 | Parameter Metadata | 13 |
| 1599.16 | Parameter Metadata | 3 |
| 1599.17 | Parameter Metadata | 1553 |
| 1599.18 | Parameter Metadata | 1551 |
| 1599.19 | Parameter Metadata | 831 |
| 1599.20 | Parameter Metadata | 832 |
| 1599.21 | Parameter Metadata | 918 |
| 1599.22 | Parameter Metadata | 1549 |
| 1599.23 | Parameter Metadata | 1560 |
| 1599.24 | Parameter Metadata | 1561 |
| 1599.25 | Parameter Metadata | 2 |
| 1599.26 | Parameter Metadata | 1576 |
| 1599.27 | Parameter Metadata | 1577 |
| 1599.28 | Parameter Metadata | 60 |
| 1599.29 | Parameter Metadata | 61 |
| 1599.30 | Parameter Metadata | 1694 |
| 1599.31 | Parameter Metadata | 0 |
| 1599.32 | Parameter Metadata | 0 |
| 1599.33 | Parameter Metadata | 0 |
| 1599.34 | Parameter Metadata | 0 |

Figure 6. Visual representation of parameters for drive identification in MCT 10.

**Table 1. Drive identification values**

| Value of F | Drive Family |
|---|---|
| 0 | P600 |
| 1 | P400 |
| 2 | MCD5xx |
| F does not exist | VLT |

The first thing to do when trying to identify a drive is to read the parameter number 1598. Its response is a string containing a series of information about the drive [9].

**Table 2. Drive responses during identification**

| Drive | Information string |
|---|---|
| P400 Mark I FC-102 | "F=1;D=FC-102;B=FC-102;L=FC-102" |
| P400 Mark II FC-302 | "F=1;D=FC-302;B=FC-302;L=FC-302;K=2" |
| FC-051 | "F=0;D=FC-051;B=FC-051;R=0270;W=37;V=220;P=1" |
| VLT 5000 | No string is retrieved because parameter 1598 does not exist. |

Old P400 drives do not have parameter 1598. In this way, the success of the identification procedure is not affected if the parameter reading fails. The response of parameter 1598 for P400 drive family does not contain as much information as, for example, for FC-051. After reading parameter 1598, parameter group 1599 with array index 22 is also read [9].

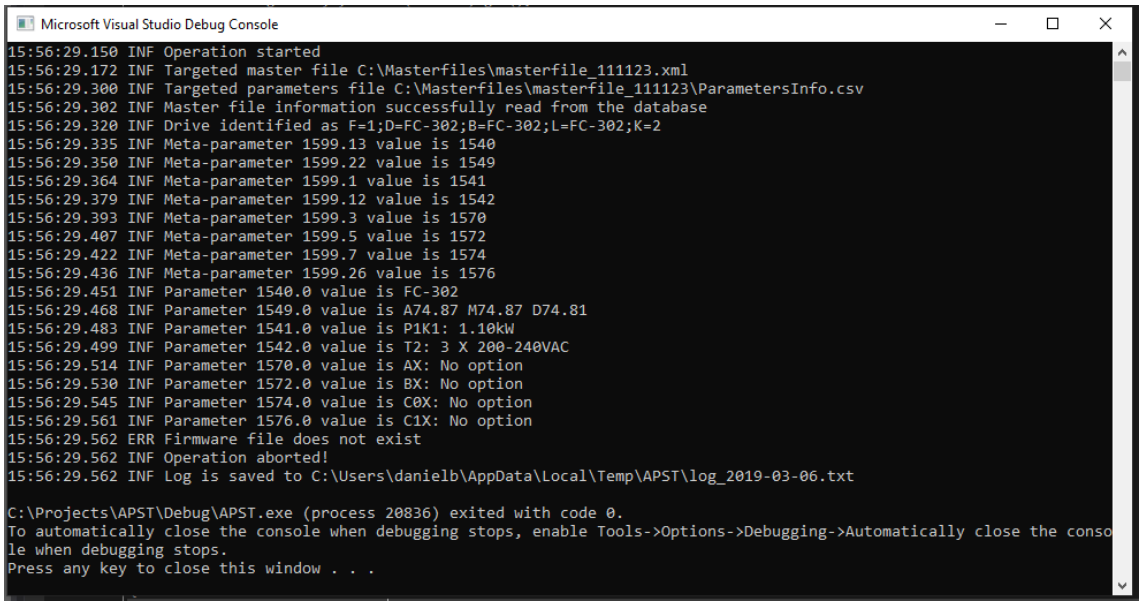



Figure 7. Example of a successful drive identification in APST.

Currently, APST only supports drives of P400 drive family (both Mark I and Mark II).

### 2.1.4 Boot mode

In order to be able to upgrade the firmware of a drive, it must be sent into the boot mode (for example, using MCT 10) if it was in the normal mode. In the normal mode, it is impossible to erase and flash the drive flash memory. Besides that, the firmware upgrade can be performed only in the boot mode for safety reasons (e.g., not to damage running motors).

### 2.1.5 Firmware file validation

In addition to what was described in the subchapter 2.1.1, OSE file can also contain validation sections. Validation data is used to ensure that the selected file is compatible with the drive that the user is trying to flash. There can be several validation sections in a single OSE file, all of which must match. Although the user can force flashing in MCT 10, given sufficient permissions, APST was designed that way that there are no user interactions, thus forcing cannot be performed. MCT 10 and APST support two types of validation sections – mask validators and token validators [8].

Mask validators read string parameter with a given number from the drive. They then match the parameter's value to a reference mask (also given in the OSE file), skipping '*'

characters in the mask. In practice, they are used to compare the drive's type code against a given mask to ensure that e.g. drive type matches [8].

Token validators use drive parameter 1598 and the value of that parameter is a semicolon-separated list of tokens, each token being in the format "Key=Value". E.g. the value of parameter 1598 on FC-102 drive could be „F=1;D=FC-102;B=FC-102;L=FC-102". Token validator checks if the value of a specific token equals to a reference value stored within the OSE file. The main purpose of the token validator was to differentiate between P618 and P619 variants of the FC-101 drive [8].

### 2.1.6 Firmware file scripts execution

The OSE file handling in APST is very similar to the one used in MCT 10 (it is crucial that APST and MCT 10 should essentially upgrade drives in the same way) and consists of three main classes.

First, the "DriveFlasher" class that handles the common functionality and is subclassed for specific drive families such as P400. Second, the "OsxScriptParser" class that parses and executes the scripts within OSE files. And third, the "DDCommWrapper" class that is used for communicating with the drive [8].

When flashing a drive, DriveFlasher and its subclass load the firmware OSE file, validate it and then proceed to execute each of the firmware sections. For every firmware section, first, the script is executed and then the firmware, if any, is sent to the drive [8].

The main responsibility of the DriveFlasher subclasses is logic for flashing binary files.

DDCommWrapper is the class used to send arbitrary data to the drive in the boot mode. It contains utility methods, e.g. to switch the drive to the boot or normal mode, nevertheless the two most important methods are the ones used to send an arbitrary command to the drive and the ones used to transfer data to the drive over Xmodem and Ymodem protocols [8].

To send an arbitrary command to the drive, the higher-level code supplies the command as well as optional prompt and timeout. The command is sent to the drive and then we wait for the drive's reply. The reply is read until the prompt text is encountered or until a timeout occurs. The prompt text defaults to `Redboot>` but can be customized by the higher-level code and the reply is different from `Redboot>` when a drive's reply is expected during data transfer [8].

### 2.1.7 Data transfer

To transfer packets of data to a drive APST uses the Ymodem protocol, which is essentially Xmodem-1K (i.e. Xmodem CRC with 1024-byte packets [10]), however, it is very similar to the conventional Xmodem/Ymodem implementation and uses the same notations (table 4), for example provided by Massachusetts Institute of Technology [11].

**Table 3. Packet structure for data transfer using Ymodem in APST.**

| Byte 1 | Byte 2 | Byte 3 | Bytes 4 - 1027 | Bytes 1028 - 1029 |
|---|---|---|---|---|
| Header | Packet number | Number of packets left | Payload (packet data) | 16-bit CRC |

**Table 4. Notations used in the protocol implementation.**

| Symbol | Description | Value |
|---|---|---|
| SOH | Start of Header | 0x01 |
| EOT | End of Transmission | 0x04 |
| ACK | Acknowledge | 0x06 |
| NAK | Not Acknowledge | 0x15 |
| ETB | End of Transmission Block | 0x17 |
| CAN | Cancel (Force receiver to start sending C's) | 0x18 |
| C | ASCII “C” | 0x43 |

The receiver starts by sending an ASCII “C” (0x43) character to the sender indicating it wishes to use the CRC method of block validating. After sending the initial “C” the receiver waits for either a 3 second time out or until a buffer full flag is set. If the receiver is timed out, then another “C” is sent to the sender and the 3 second time out starts again [11]. This process continues until the receiver receives a complete 1029-byte packet.

Indeed, using Log Visualizer (internal proprietary tool) it is possible to observe that after running a command `dtm field_flash_erase -d 2` to erase the previous content on a drive, drive sends `Redboot>` and then APST initiates the data transfer. The drive acknowledges that and sends an ASCII symbol "C". The data transmission starts and once the packet is sent, the drive sends ACK (0x06) notifying readiness to receive the next one.

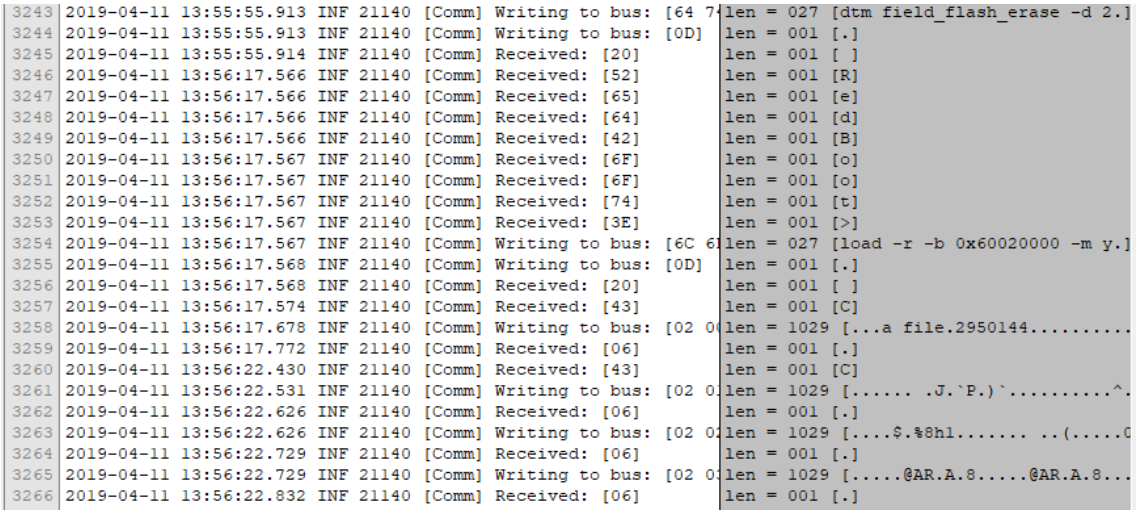



Figure 8. Communication log of drive flashing initialization

When the flashing process is completed, APST sends the `dtm mode_change` command to switch the drive to the normal mode. After that, baud rate, parity bit and stop bits are set to default values since during flashing the set up is different (e.g., the baud rate is set to 115200 bd/s).

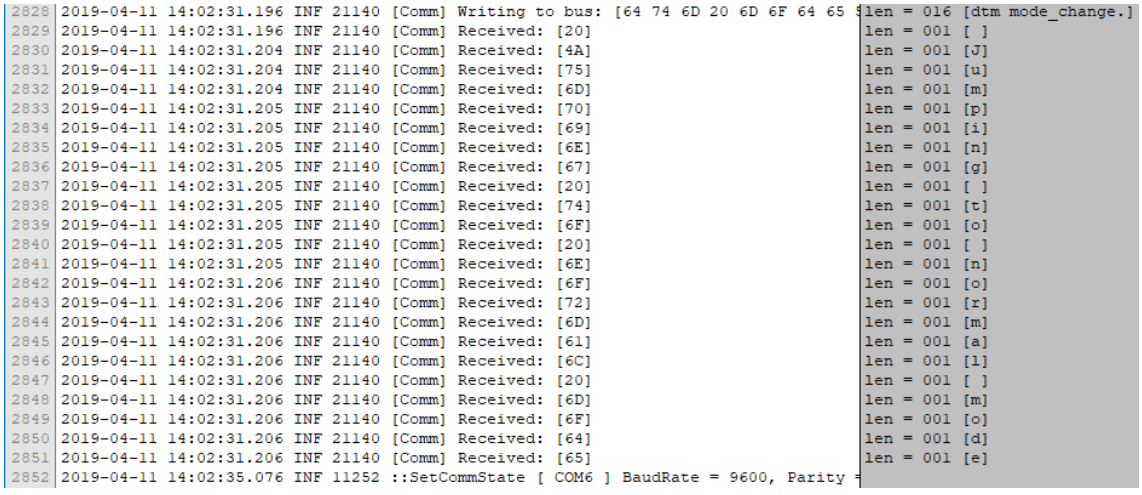



Figure 9. Communication log after drive flashing.

## 2.2 Test scenarios

The subchapter provides some of the test cases on which the APST flashing feature has been tested.

1. Trying to flash an unsupported drive family (e.g., P600)

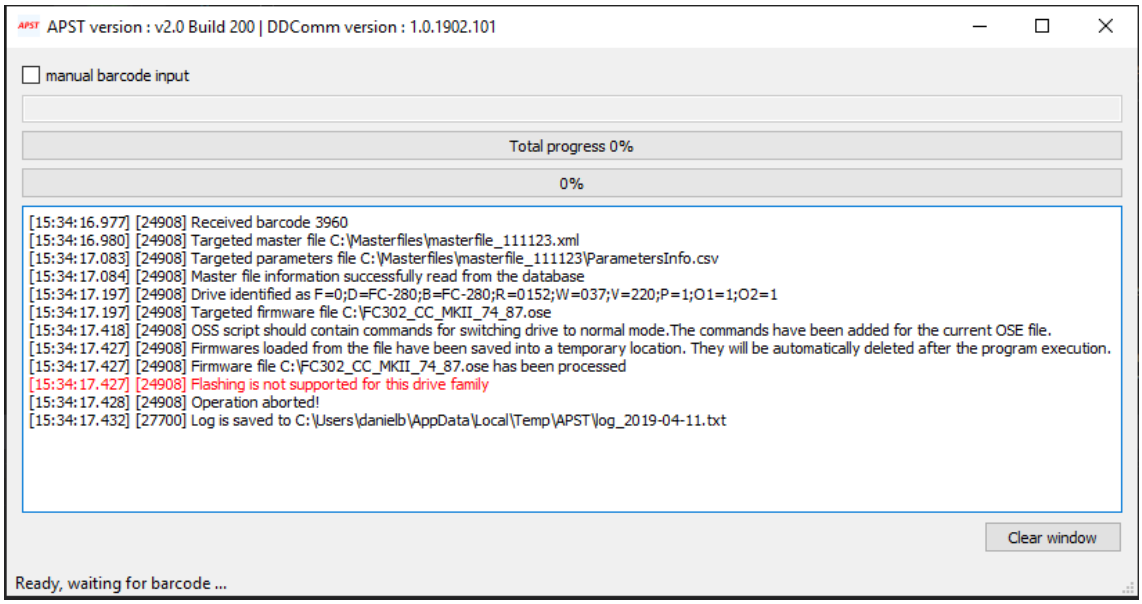



Figure 10. Testing APST on a P600 drive.

In such a case, the drive is still correctly identified, however, APST simply aborts the execution as the drive family is not supported just yet.

2. Invalid firmware file content

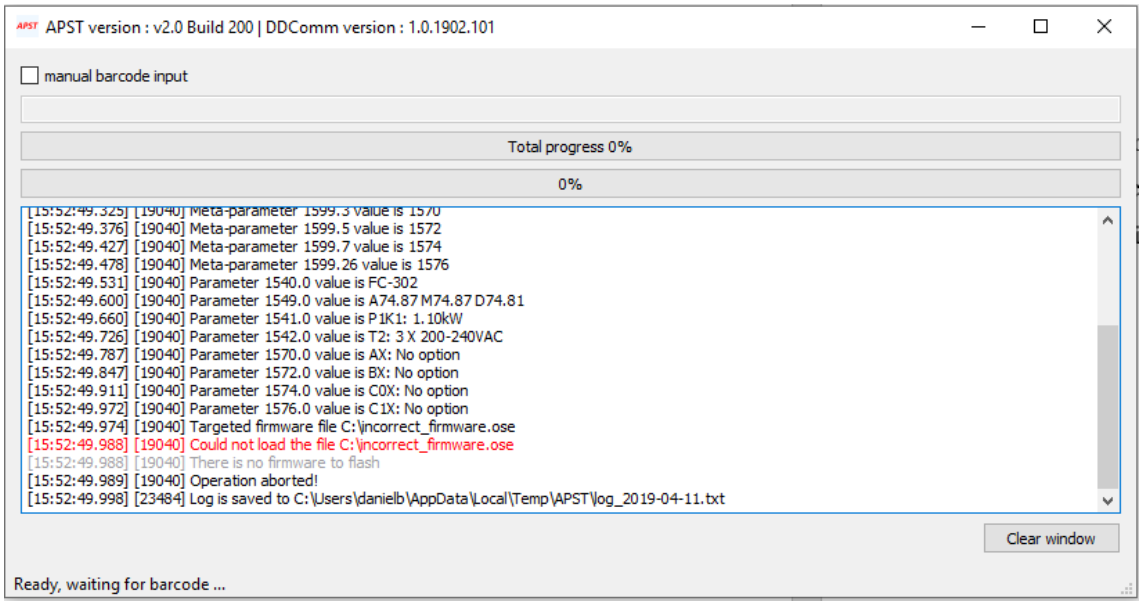



Figure 11. Testing APST on an incorrect firmware file.

The given firmware file does not pass the validation stage. APST aborts the execution.

3. Trying to access the serial bus that is being used

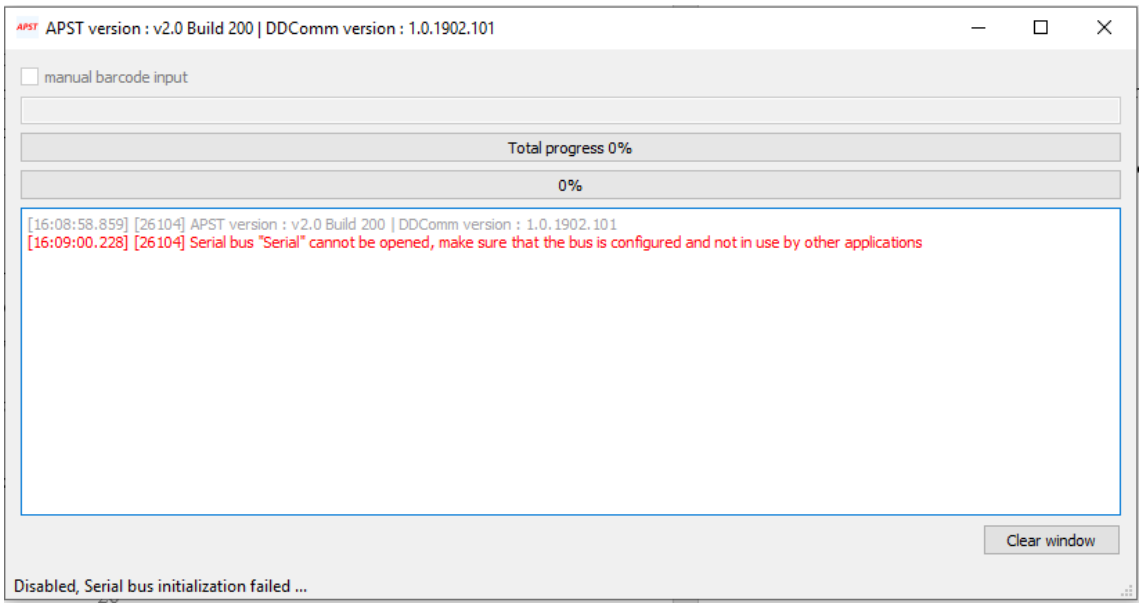



Figure 12. Testing APST on an in-use serial bus.

In such a case, APST does not access the busy bus, since it most probably will lead to synchronization and access violation errors.

## 2.3 Future work

The main part of APST development for drive firmware upgrade has been successfully implemented and, in the future, it is planned to include such features and improvements in APST:

1. Add support of Universal Serial Bus (USB) for speed enhancing (the difference can be up to 10 times compared to drive upgrade using a serial port through RS-232/485 converter (e.g., ADAM-4561 converter).

2. Implementation of the progress bar. Currently, the progress bar in GUI is not implemented, thus the end user can only predict when the flashing will be completed (in average, it takes around 9-10 minutes over the serial port and Ymodem protocol) by looking at the flash log.

3. Currently, APST is used for automation of Danfoss drives setup (e.g., specific parameters values for controlling motors) in manufacturing companies located in Europe that use P400 drives, though as APST will progress, support of the P600 drive family most probably will be relevant.

   The P600 product line started as a shortened version of P400 drives which was meant to be sold only in China. Each P600 drive has to some extent a major deficiency compared to P400 drives so that P600 would not compete with P400 drives.

4. Presently, it still happens that sometimes the drive's port settings upon flashing are not restored correctly (parity bit and baud rate) making it not possible to immediately write binary files and setup settings (parameters values) after flashing. This case needs to be thoroughly investigated and fixed before being released into production.

# 3 Operating with APST

The chapter represents an overview of APST overall, in particular, its features and discusses the main points to remember while using APST (including its configuration).

The flashing feature in APST is a one step forward towards facilitating work of technicians on the production line to not use MCT 10 for such use cases. Furthermore, at the end of the chapter, we present the actual use cases of APST and means for debugging in case errors take place.

## 3.1 Installation

For the tool to work properly, the latest version of MCT 10 must be installed on the computer. MCT 10 will install a communication module that is required for communicating with the drives [7].

APST installer is provided as a self-extracting executable. After running APST_Installer.exe, it will ask for a location where to extract the needed files.

Upon the installation, a new folder called “APST” will be created in the chosen location with the installed software.

## 3.2 APST Configuration

First, the serial bus must be correctly configured. This must be performed from MCT 10 user interface.
Typical settings are related to a communication port (COM) and baud rate [7]:

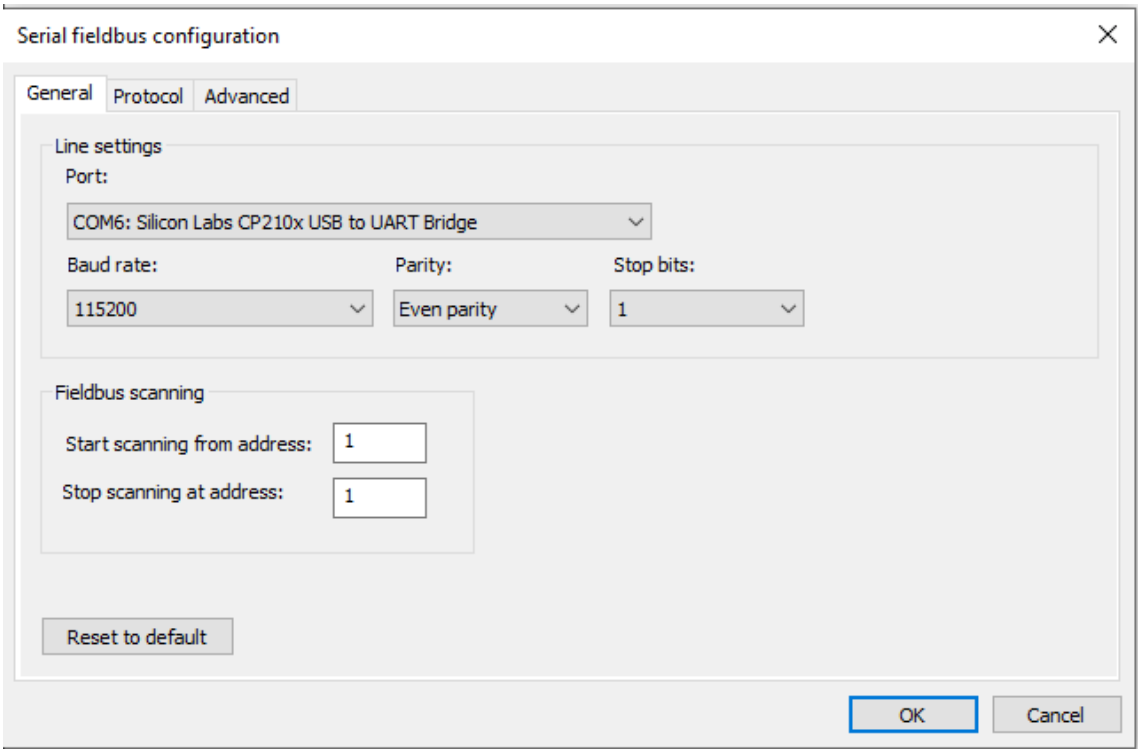



Figure 13. Serial fieldbus configuration.

After the serial bus has been configured, MCT 10 must be closed as any other application that might keep the COM port locked. The next step is to configure the serial bus name that APST shall use.

In the same directory where APST is installed, there must also be present a configuration file called **config.xml**. The file contains three settings [7]:

1. `<DbDirectoryPath>` is meant for the Desktop version of APST. It specifies the location where APST should search for master files. This setting can be ignored when APST is used as a command line tool.
2. `<SerialBusName>` is a mandatory configuration. The name defined in here must match with the name defined in MCT 10 for the serial bus that APST shall use.
3. `<P400DriveGarbageCollectionEnabled>` is an optional configuration. The default value is “True”, which means that MCT 10 will perform garbage collection before writing binary files. With a value “False” set, garbage-collection is skipped. It is recommended to use “False” only for testing purposes.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <DbDirectoryPath>C:\Masterfiles</DbDirectoryPath>
    <SerialBusName>Serial</SerialBusName>
</Configuration>
```

Figure 14. Example of the config.xml file.

## 3.3 APST commands

The tool can run as a command line. For a list of supported commands, a help command must be called: apst.exe --help:

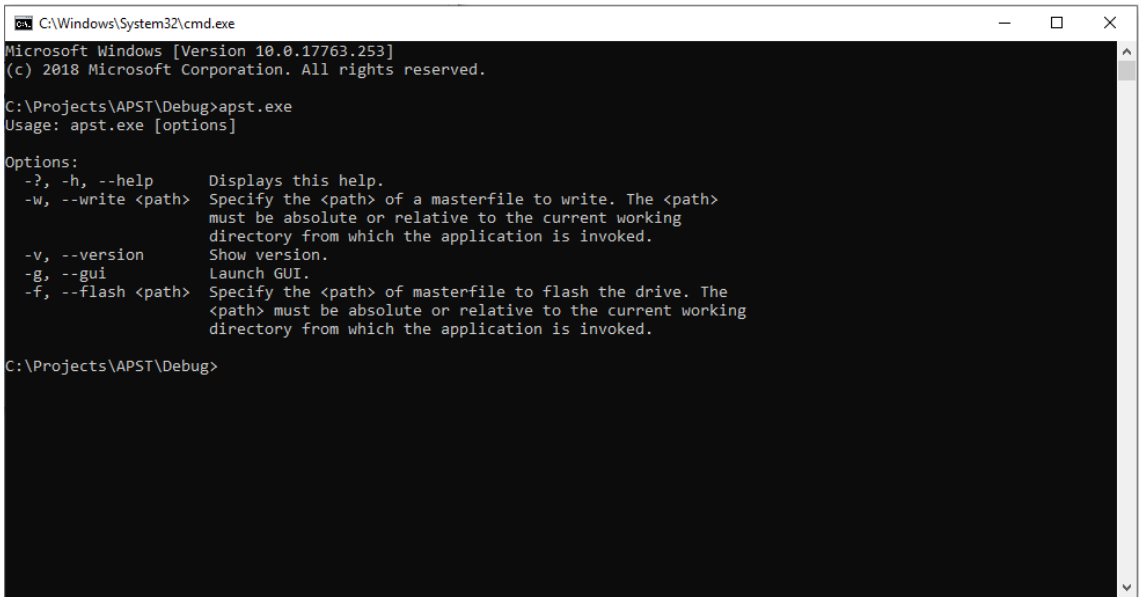



Figure 15. APST command line interface.

The following command line parameters are supported [7]:

**Table 5. List of arguments that APST supports.**

| Parameter | Description |
|---|---|
| `--write, -w <path>` | Specify the <path> of a master file to write. The <path> must be absolute or relative to the current working directory from which the application is invoked. |
| `--flash, -f <path>` | Specify the <path> of a master file to flash. The <path> must be absolute or relative to the current working directory from which the application is invoked. After flashing is completed, parameters are written to the drive (identical to --write/-w) if there are any to be written. |
| `--help, -h, -?` | Displays help. |
| `--version, -v` | Displays tool and communication module version number. |
| `--gui, -g` | Initiates GUI to be shown. Can be used for scanning a barcode. |

### 3.3.1 The write command

The write command is used to perform a commissioning operation to the drive, commissioning operation that consists in writing parameter values to drive and files. The write command takes as input a so-called Masterfile. A Masterfile is a file that tells APST where to find all the information needed to perform commissioning to a drive.
The information needed by APST is collected in a series of files (in UTF-8 encoding), and the Masterfile simply points APST to the directory where those information files are located. Masterfile format is XML [7].

The information files that the Masterfile points to consist of [7]:

- Information about writing sequence (in what order parameters shall be written), parameters number, datatype and eventually scaling factor. This information is collected in an XML file in a format specified in the Masterfile subchapter.
- Information about parameter value to be written, and what setup number to write. This information is collected in a CSV file in a format specified in the Drive Information subchapter.

### 3.3.2 The flash command

The flash command is used to perform an upgrade of firmware that is currently installed in the target drive. A detailed description of the flash command algorithm is outlined in APST algorithm subchapter.

### 3.3.3 The help command

The help command simply prints out in the console possible commands with a respective syntax to operate with APST.

### 3.3.4 The version command

The version command outputs the version and build number of APST and the version of the communication module that is installed.

### 3.3.5 Using APST from the console

From a console window, APST commands can be called as the following example shows:

*apst.exe --write "C:\masterfiles\mydrive.xml"*

In case of the write command, APST logs to the console window (and to the log file as well) the progress and the information about the current operation. If the operation succeeds, the tool returns 0 (zero). In case of any failure, the tool returns 1 (one).
The return code can be read by typing:
*Echo %ERRORLEVEL%*

### 3.3.6 Using APST from a scripting language

APST commands can be executed from a scripting language [7]. If the operation succeeds, the tool returns 0 (zero). In case of any failure, the tool returns 1 (one).
Example of usage from a .bat file:

```
call C:\apst\apst.exe --write "C:\masterfiles\masterfile_m.xml"

IF %ERRORLEVEL% EQU 0 (Echo Succeeded) ELSE (Echo Failed)
```

Figure 16. Using APST from a scripting language.

## 3.4 Configuring APST

The tool loads configuration from the **config.xml** file, If the file is missing, APST gives the following error:

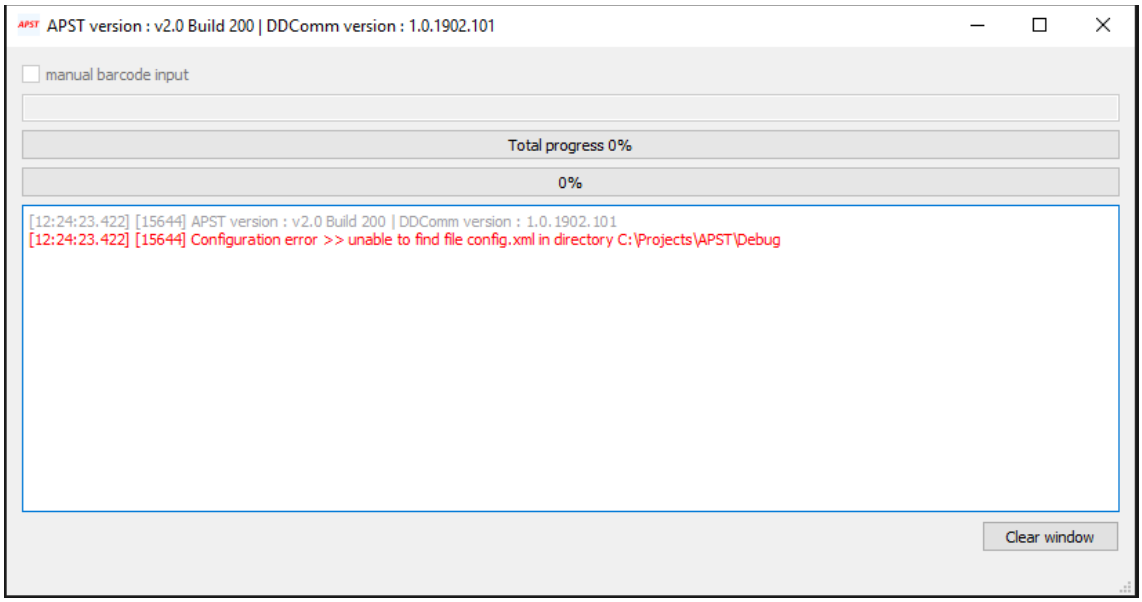



Figure 17. Example of the configuration error.

The configuration file and the executable must be in the same directory. See table 6 for XML tags description [7].

**Table 6. Configuration file XML tags description.**

| XML tag | Description | Default value |
|---|---|---|
| `DbDirectoryPath` | Masterfile database path when the tool is launched with a `-g/--gui` argument | - |
| `SerialBusName` | Name of the serial bus | Serial |
| `P400DriveGarbageCollectionEnabled` | With value False garbage-collection before writing the binary files is skipped. It is recommended to use skipping only for testing purposes. | True |

```
<?xml version="1.0" encoding="UTF-8"?>
	<Configuration>
		<DbDirectoryPath>C:\Temp\masterfiles</DbDirectoryPath>
		<SerialBusName>Serial_2</SerialBusName>
		<P400DriveGarbageCollectionEnabled>False</P400DriveGarbageCollec
	tionEnabled>
	</Configuration>
```

Figure 18. The configuration file example.

## 3.5 Master file

To use APST, it is crucial to understand the content of master files and what XML tag corresponds to what setting. The master file itself is the file that connects information from ERP systems to Danfoss Drives project information. In this way it contains information about parameters and files to be written to a drive. The master file has an XML format (UTF-8 encoding) [7].

**Table 7. Master file XML tags description.**

| XML tag | Description |
|---|---|
| `DriveInfoFileName` | Path to the DriveInfo XML file. This file is needed for the tool to recognize a correct write order and parameter information (id, datatype and scale factor). If a full path is not used, then the specified path will be considered relative to the master file location. |
| `ParameterFileName` | Path to the ParametersInfo CSV file. The file where parameters to be written are described. If a full path is not used, then the specified path will be considered relative to the master file location. |
| `Barcode` | Barcode ID. Links the Masterfile to the production system. |
| `Customer` | Information about the customer. Optional. |
| `Order Number` | Information about the order. Optional. |
| `IdentificationParameters` | Contains information about the drive. |
| `BinaryFiles` | Contains paths to binary files. |
| `BinaryFile` | Binary file path. If a full path is not used, the file must be located at the same root as the Masterfile. Optional. |
| `FirmwareFile` | Firmware file (.OSE) path. Optional. |

```
<?xml version="1.0" encoding="UTF-8"?>
<MasterFile>
	<Barcode>1925177829</Barcode>
	<DriveInfoFileName>masterfile_name\DriveInfo.xml</DriveInfoFileName>
	<ParametersFileName>masterfile_name\ParametersInfo.csv</ParametersFileName>
	<BinaryFiles>
		<BinaryFile>masterfile_name\pump0068.bin</BinaryFile>
		<BinaryFile>masterfile_name\my.splash</BinaryFile>
		<BinaryFile>masterfile_name\vigala.sas</BinaryFile>
		<BinaryFile>masterfile_name\english.lng</BinaryFile>
	</BinaryFiles>
	<FirmwareFile>C:\FC202_2_63.ose</FirmwareFile>
</MasterFile>
```

Figure 19. Master file example.

## 3.6 DriveInfo XML file

The DriveInfo XML file is used as a drive descriptor. It contains information about MCT 10 write order of parameters (some parameters depend on other) and necessary information about parameters (ID, scaling factor and data type) [7].

**Table 8. DriveInfo file description.**

| Field Name | Description |
|---|---|
| `ParameterUpdateSequence` | Information about MCT 10 write order: order on how `Parameters` will have to be written. |
| `Parameters` | Information about parameters to be written. |
| `Parameter` | Parameter information: Must contain `ParamNumber`, `ScalingFactor` and `DataType`. |
| Tag Name | Description |
| `ReadAndCompareTimeOutMSec` | Upon writing the value, APST has to wait up for a specific timeout until the value is changed in the drive. |
| `DataType` | Data type of parameter. |

```
<?xml version="1.0" encoding="UTF-8"?>
<DriveInfoFile>
    <ParameterUpdateSequence>
        <Item ParamNumber="12" Setup="1"/>
        <Item ParamNumber="1459" Setup="1"/>
        <Item ParamNumber="847" Setup="4"/>
        <Item ParamNumber="847" Setup="3"/>
        <Item ParamNumber="847" Setup="2"/>
        <Item ParamNumber="847" Setup="1"/>
    </ParameterUpdateSequence>
    <Parameters>
        <Parameter ParamNumber="12" ScalingFactor="0" DataType="DT_UNSIGNED8"
ReadAndCompareTimeOutMSec="20000"/>
        <Parameter ParamNumber="847" ScalingFactor="0"
DataType="DT_UNSIGNED16"/>
        <Parameter ParamNumber="1559" ScalingFactor="0"
DataType="DT_VISIBLE_STRING"/>
    </Parameters>
</DriveInfoFile>
```

Figure 20. Example of DriveInfo.xml file.

## 3.7 ParametersInfo CSV file

The ParametersInfo file contains information about parameters as they are presented in MCT 10. Parameters are in CSV format. The CSV file is in UTF-8 encoding [7].

**Table 9. ParameterInfo file description.**

| Meaning | Additional info | Char | Example |
|---|---|---|---|
| Comment | | # | field_1;field_2<br>#this line is commented |
| Decimal delimiter | | . or , | 1.1 or 1,1 to support different locales. Typically .(dot) or ,(comma) |
| Empty | | | field_1;;field_3 |
| Field delimiter | The user must specify delimiter using separator specifications as described below. Delimiter specified by the user must not exceed one character in length. | ; | field_1;field_2;field_3 |

### 3.7.1 Using ParametersInfo file

For using the content of the ParametersInfo file, separator and date-time format fields need to be present. More information about date-time formats is available in Appendix 1.

Separator and date-time format fields must be placed at the beginning of the CSV file. Supported fields (all fields are mandatory) [7]:

**$FIELD_SEPARATOR**
**$DECIMAL_SEPARATOR**
**$THOUSANDS_SEPARATOR**
**$DATETIME_FORMAT**
**$TIME_FORMAT**
**$DATETIME_LANGUAGE**

The field must be followed by = (equal) char and a proper value. Parameter values must be in the same format as in MCT 10 parameters grid, except choice-list parameters which must have raw value. For example, for writing the language to be English value "0" is used.

```
$FIELD_SEPARATOR=;
$DECIMAL_SEPARATOR=.
$THOUSANDS_SEPARATOR=,
$DATETIME_FORMAT=M/d/yyyy h:mm:ss AP
$TIME_FORMAT=h:mm:ss AP
$DATETIME_LANGUAGE=33

#parameter id;setup 1 value;setup 2 value;setup 3 value;setup 4 value
P001;1 # 1=Deutsch
P460.15;1;0;0;1
P1300;;;20.00
```

Figure 21. Example of the ParametersInfo.xml file.

If the used month representation in the date-time format is MMM or MMM, then $DATETIME_LANGUAGE field is used by APST to determine the language [7].

Example:

English/United States 1

Danish 29

English 31

German 42

Qt documentation was used as a reference to define the $DATETIME_LANGUAGE values for the enumeration.

### 3.7.2 Logging

APST commands were developed that way that they are featured with a logging mechanism that simply outputs the information in the corresponding console window,

which is extremely helpful for an operator on a production line and for the development purposes (e.g. debugging). All the information that is seen in the window logs also appears in a text file. The text file is named "log_currentdate.txt", where the current date is represented using the format YYYY-MM-DD. For example, a log file can be named as follows [7]:

log_2019-04-06.txt

A new log file is created once a day. Otherwise, it is appended during that day. Every line in the log file starts with a timestamp represented using the format HH:MM:SS, for example:

*12:24:37.027 State >> Parameter 413.0 is written successfully, duration: 16 msec. Value is changed to 0*

The log file is saved to the folder named *app_name* which is placed to the TEMP folder. For example, a saved log file path could look like this (Windows 7 operating system):

*C:\Users\myUserName\AppData\Local\Temp\APST\log_2019-04-06.txt*

An example log file after flashing a drive can be seen in Appendix 2.

## 3.8 Use cases

Before proceeding to the use cases, technical details need to be defined:

1. Production file represents two files: Master file (CSV or XML) and ParametersInfo file (CSV). See 3.5 and 3.7 subchapters for their content description.

2. User must install APST with a CD key and license. If MCT 10 is already installed, the tool will automatically use this information.

The provided use cases assume next prerequisites [7]:

- The customer must use the serial fieldbus between a production PC and the drive.

- The production PC can be either a Programmable Logic Controller (PLC) or a PC, nonetheless, it must be Windows based and as a minimum have DDComm

(optionally MCT 10) and the Automation Production Setup Tool installed and running.

- The production PC must either have access to the local network (to access the production file: CSV/XML and the ERP system of the customer) or the files must be stored locally on the production PC.

Once the prerequisites are fulfilled, the engineer must do the following [7]:

1. Create a project in MCT 10 (or use an existing), which contains the specific drive and parameters (one project for each drive) or alternatively, the engineer can create the Master file (CSV or XML) from a template and then must add, for example, the link to the firmware file.

2. Via MCT 10 the engineer converts the project file to the production file and stores it on the local network.

3. The engineer opens the Master File (CSV or XML) in a text editor and adds the customer's specific barcode in the corresponding tag. This is done to link the Master file to the production system. Finally, the engineer stores the production file on the local network or on the production PC.

4. On the production line, the technician scans the barcode of the drive (see figure 22 for a drive example) using a barcode scanner, which is connected to a production PC and that way connects the specific drive to the production PC.

5. The technician powers up the drive (APST is in the ready mode), and when the drive is ready and can be identified, APST validates and starts flashing and/or writing the files immediately (see chapter 2 for the exact algorithm details).

6. APST selects the correct Master file and validates that all linked files are available and starts to flash and/or write the files to the drive.

7. If somehow the drive is in a non-working state (e.g. the drive is in the boot mode, not started up and etc.), then APST displays an error message and stores the fault information in a log .txt file.

8. The technician acknowledges the error and program goes into the ready mode. The technician removes the drive from the production line and together with the fault log it is sent to the engineer.

9. The technician connects a new drive and performs the same steps described above.

10. When finished and drive is up and running again, APST validates the files by checking the parameters settings.

Figure 22. An example of a P400 drive that can be upgraded over APST.

# 4 Summary

In this thesis, the problem of the automatic drive upgrade process on a production line was addressed by developing software focusing on a human factor error-free usage with minimal user interaction by using a barcode scanner, so that the solution can substitute utilization of MCT 10.

The solution is going to be tested by quality engineers on all drives of the P400 drive family before the actual release of APST. If the solution passes all the test scenarios, APST can be used in production despite the limitation on only the P400 drive family (Mark I and Mark II).

In the “Future work” subchapter it is discussed what features are going to be implemented in the nearest future.

Thus, the aim of the thesis has been fulfilled and the questions from the motivation subchapter are considered to be positively answered.

# References

[1] Danfoss A/S, “VLT® AutomationDrive FC 301 / FC 302 documentation” (in Russian), http://drives.danfoss.ru/products/vlt/low-voltage-drives/vlt-automation-drive-fc-301-302/#/ (accessed 29.04.2019)

[2] Danfoss A/S, “Operating Guide VLT® Motion Control Tools MCT 10 Set-up Software”

[3] Proekspert company, https://proekspert.ee/ (accessed 30.04.2019)

[4] Internal documentation, “DDComm Architecture overview”, https://intra.proekspert.ee/wiki/display/DPCT/Architecture+Overview, 2010 (accessed 30.04.2019)

[5] The Qt company, “About Qt”, https://wiki.qt.io/About_Qt (accessed 29.04.2019)

[6] C++ boost libraries, https://www.boost.org/ (accessed 29.04.2019)

[7] Danfoss A/S, “Automation Production Setup Tool. Practical Guide and User Manual”, 2015

[8] Internal documentation, “DSU and OSE format”, https://intra.proekspert.ee/wiki/display/DPCT/DSU+and+OSE+format, 2011 (acessed 30.04.2019)

[9] Internal documentation “Drive identification”, https://intra.proekspert.ee/wiki/display/DPCT/Drive+Identification, 2018 (accessed 30.04.2019)

[10] Portland State University, “The Guide”, 1993, http://web.cecs.pdx.edu/~rootd/catdoc/guide/TheGuide_226.html (accessed 29.04.2019)

[11] Massachusetts Institute of Technology, “Xmodem protocol with CRC”, http://web.mit.edu/6.115/www/amulet/xmodem.htm (accessed 29.04.2019)

[12] The Qt company, “Locale class. Date-time formats in Qt”, https://doc.qt.io/archives/qt-4.8/qlocale.html#Language-enum (accessed 29.04.2019)

# Appendix 1 Date-time formats

Qt documentation was used as a reference to define the date-time formats (see table 10, 11) [12].

**Table 10. Date-time formats in APST.**

| Expression | Output |
|---|---|
| d | The day as number without a leading zero (1 to 31) |
| dd | The day as number with a leading zero (01 to 31) |
| M | The month as number without a leading zero (1-12) |
| MM | The month as number with a leading zero (01-12) |
| MMM | The abbreviated localized month name (e.g. 'Jan' to 'Dec') in the $DATETIME_LANGUAGE defined language |
| MMMM | The long-localized month name (e.g. 'January' to 'December') in the $DATETIME_LANGUAGE defined language. |
| yy | The year as a two-digit number (00-99):<br>69 = 2069,<br>70 = 1970,<br>71 = 1971 |
| yyyy | The year as a four-digit number. |

**Table 11. Date-time formats in APST.**

| Expression | Output |
|---|---|
| h | The hour without a leading zero (0 to 23 or 1 to 12 if AM/PM display) |
| hh | The hour with a leading zero (00 to 23 or 01 to 12 if AM/PM display) |
| m | The minute without a leading zero (0 to 59) |
| mm | The minute with a leading zero (00 to 59) |
| s | The second without a leading zero (0 to 59) |
| ss | The second with a leading zero (00 to 59) |
| zzz | The milliseconds with leading zeroes (000 to 999) |
| AP | Use AM/PM display. AP will be replaced by either "AM" or "PM" |
| ap | Use am/pm display. ap will be replaced by either "am" or "pm |

**Table 12. Example formats.**

| Format | Result |
|---|---|
| dd.MM.yyyy MMMM d yy | 21.05.2001 May 21 01 |
| hh:mm:ss.zzz | 14:13:09.042 |
| h:m:s ap | 2:13:9 pm |

# Appendix 2 Log file example

```
11:49:03.230 INF 37340 GUI application started
11:49:03.256 DBG 37340 APST version : v2.0 Build 200 | DDComm version :
1.0.1902.101
11:49:03.382 INF 37340 GUI application initialized successfully
11:49:03.382 INF 37340 Database directory set to C:\Masterfiles
11:52:40.494 INF 37340 Operation started
11:52:40.501 INF 31980 Received barcode 9326598264
11:52:40.504 INF 31980 Targeted master file
C:\Masterfiles\masterfile_111123.xml
11:52:40.603 INF 31980 Targeted parameters file
C:\Masterfiles\masterfile_111123\ParametersInfo.csv
11:52:40.606 INF 31980 Master file information successfully read from the
database
11:52:40.624 INF 31980 Drive identified as F=1;D=FC-302;B=FC-302;L=FC-302;K=2
11:52:40.639 INF 31980 Meta-parameter 1599.13 value is 1540
11:52:40.654 INF 31980 Meta-parameter 1599.22 value is 1549
11:52:40.669 INF 31980 Meta-parameter 1599.1 value is 1541
11:52:40.684 INF 31980 Meta-parameter 1599.12 value is 1542
11:52:40.699 INF 31980 Meta-parameter 1599.3 value is 1570
11:52:40.714 INF 31980 Meta-parameter 1599.5 value is 1572
11:52:40.729 INF 31980 Meta-parameter 1599.7 value is 1574
11:52:40.744 INF 31980 Meta-parameter 1599.26 value is 1576
11:52:40.758 INF 31980 Parameter 1540.0 value is FC-302
11:52:40.775 INF 31980 Parameter 1549.0 value is A74.87 M74.87 D74.81
11:52:40.790 INF 31980 Parameter 1541.0 value is P1K1: 1.10kW
11:52:40.806 INF 31980 Parameter 1542.0 value is T2: 3 X 200-240VAC
11:52:40.821 INF 31980 Parameter 1570.0 value is AX: No option
11:52:40.837 INF 31980 Parameter 1572.0 value is BX: No option
11:52:40.852 INF 31980 Parameter 1574.0 value is C0X: No option
11:52:40.867 INF 31980 Parameter 1576.0 value is C1X: No option
11:52:40.868 INF 31980 Targeted firmware file C:\FC302_CC_MKII_53_00.ose
11:52:41.126 INF 31980 OSS script should contain commands for switching drive
to normal mode.The commands have been added for the current OSE file.
11:52:41.134 INF 31980 Firmwares loaded from the file have been saved into a
temporary location. They will be automatically deleted after the program
execution.
11:52:41.134 INF 31980 Firmware file C:\FC302_CC_MKII_53_00.ose has been
processed
11:52:41.134 DBG 31980 Send drive to boot mode
11:52:41.257 DBG 31980 Apply boot mode configuration for the bus
11:52:41.783 DBG 31980 Send command: \r
11:52:43.566 DBG 31980 Received RedBoot
11:52:43.566 INF 31980 DTM version 2.5
11:52:43.566 INF 31980 Starting software upgrade
11:52:43.566 INF 31980 Initializing
11:52:43.566 INF 31980 Flashing from file C:\FC302_CC_MKII_53_00.ose
11:52:43.816 INF 31980 Read DTM version
11:52:43.820 INF 31980 DTM version: 2.05
11:52:43.820 INF 31980 Validate software file
```

```
11:52:43.820 INF 31980 Executing script #1
11:52:43.820 INF 31980 Sending command to the drive: 'dtm readmem -a
0x1fff7800 -b'
11:52:43.820 DBG 31980 Send command: dtm readmem -a 0x1fff7800 -b
11:52:43.821 DBG 31980 Send command: \r
11:52:43.830 DBG 31980 Received the command
11:52:43.830 DBG 31980 Response from drive: 0xff
RedBoot>
11:52:43.830 DBG 31980 Response from drive: '0xff
'
11:52:43.830 INF 31980 CC Type check - OK
11:52:43.830 INF 31980 Sending command to the drive: 'dtm flash_size -d 2'
11:52:43.830 DBG 31980 Send command: dtm flash_size -d 2
11:52:43.831 DBG 31980 Send command: \r
11:52:43.840 DBG 31980 Received the command
11:52:43.840 DBG 31980 Response from drive:  8192 kB
RedBoot>
11:52:43.840 DBG 31980 Response from drive: ' 8192 kB
'
11:52:43.840 INF 31980 CC Flash check - OK
11:52:43.840 INF 31980 CC Flash erase - started...
11:52:43.840 INF 31980 Sending command to the drive: 'dtm field_flash_erase -
d 2'
11:52:43.840 DBG 31980 Send command: dtm field_flash_erase -d 2
11:52:43.840 DBG 31980 Send command: \r
11:53:05.310 DBG 31980 Received the command
11:53:05.311 DBG 31980 Response from drive:  RedBoot>
11:53:05.311 DBG 31980 Response from drive: ' '
11:53:05.311 INF 31980 CC Flash erase - OK
11:53:05.311 INF 31980 Sending command to the drive: 'load -r -b 0x60020000 -
m y'
11:53:05.311 DBG 31980 Send command: load -r -b 0x60020000 -m y
11:53:05.312 DBG 31980 Send command: \r
11:53:05.319 INF 31980 Sending 2950144 bytes of data to the drive using y-
modem
11:53:05.320 DBG 31980 Drive response: 67
11:53:05.421 DBG 31980 Sending header:
11:53:10.175 DBG 31980 Drive response: 67
11:53:10.276 DBG 31980 Initiating transmission:
11:53:10.276 DBG 31980 Packet 1
11:53:10.371 DBG 31980 Packet 2
…
…
Packet n
…
11:58:06.595 INF 31980 Executing script #2
11:58:06.596 INF 31980 Sending command to the drive: 'load -r -b 0x60764400 -
m y'
11:58:06.596 DBG 31980 Send command: load -r -b 0x60764400 -m y
11:58:06.596 DBG 31980 Send command: \r
11:58:06.603 INF 31980 Sending 637952 bytes of data to the drive using y-
modem
```

```
11:58:11.760 DBG 31980 Drive response: 67
11:58:11.862 DBG 31980 Sending header:
11:58:16.614 DBG 31980 Drive response: 67
11:58:16.715 DBG 31980 Initiating transmission:
11:58:16.715 DBG 31980 Packet 1
11:58:16.810 DBG 31980 Packet 2
…
…
Packet n
…
11:59:21.562 INF 31980 CC Flash update - OK
11:59:21.563 INF 31980 Switching drive to normal mode
11:59:21.563 DBG 31980 Send drive to normal mode
11:59:21.563 DBG 31980 Send command: \r
11:59:21.570 DBG 31980 Received the command
11:59:21.570 DBG 31980 Response from drive: RedBoot>
11:59:21.570 DBG 31980 Send command: dtm write_ee -a 0 -d 0101\r
11:59:21.582 DBG 31980 Received the command
11:59:21.582 DBG 31980 Response from drive:  RedBoot>
11:59:21.582 DBG 31980 Send command: dtm mode_change\r
11:59:21.594 DBG 31980 Received the command
11:59:21.594 DBG 31980 Response from drive:  Jumping to normal mode
11:59:21.594 DBG 31980 Apply normal mode configuration for the bus
11:59:25.470 INF 31980 Flashing has been completed
11:59:25.473 INF 31980 Operation succeeded!
11:59:25.477 INF 37340 Log is saved to
C:\Users\danielb\AppData\Local\Temp\APST\log_2019-04-09.txt
```