FULL Engineering

AkzoNobel
Tomorrow's Answers Today

Universal Dispenser Communication Protocol

Interface specification: Version 1.0

October 24, 2008
R.P.C. Meeuwisse
H. van der Flier
D. van Golverdingen

You can contact Lenteq and/or AkzoNobel by e-mail (AIP@akzonobel.com) or by writing to Akzo Nobel nv, Intellectual Property, Velperweg 76, 6824 BM, Arnhem, The Netherlands.

Version Control			
Version	Date	Author	Modifications
Original interface specification	2006..2008	R.P.C. Meeuwisse	Initial Protocol Specifications
0.1 draft	February 2, 2008	H.C. van der Flier	Initial Document
0.9	April 4, 2008	H.C. van der Flier	Final draft
0.91	June 9, 2008	H.C. van der Flier	Changes from UDCP meeting May 29, 2008
1.0	August 15, 2008	E. van Noort	Layout Inclusion Legal text Exclusion Dispenser Family Codes
	August 15, 2008	H.C. van der Flier	Reference to external appendix Dispenser Family Codes included
	October 24, 2008	H.C. van der Flier	Remarks from Rene Meeuwisse were included. Punch command added CanisterContents section in the result file must now always contain all keys

TABLE OF CONTENT

Introduction

This document describes the UDCP protocol which will be used in communication between dispenser hardware/drivers on one side and point of sales (POS) dispensing software on the other. The purpose of introducing UDCP is to provide a standardized way of communication. Using UDCP, the complexity of supporting various types of dispensers will no longer have to be built into the POS software. Instead when complying with UDCP dispenser manufacturers will be guaranteed that their hardware will work with POS software that supports UDCP. UDCP stands for Universal Dispenser Communication Protocol.

Interface

1.1 Method

The UDCP interface method is based on file exchange. Files may be exchanged on one computer between the POS software and dispenser driver software; they may also be exchanged via network shares. There are no limitations as to which file system is used. The protocol layer provides for client server communication between exactly one driver/dispenser and one POS application. In this document the server refers to the POS software and client refers to the driver.

The driver must be a windows application that runs silently in the background. The POS application is responsible for starting and stopping the driver application. The only user interface allowed is the dispenser's service maintenance window (see 1.21.10 SHOWSERVICEWINDOW extension). This is also the only time the driver is allowed to take the focus. No other user interface is allowed.

Aforementioned restrictions on the user interface do not apply when the driver runs as a standalone application, when it's not launched by the server application. The driver must check the running processes to see if the server application is running. If the server application is not running the driver is allowed to provide a more elaborate user interface.

1.2 Settings

UDCP prescribes a shared windows registry key which contains settings for both dispenser drivers and POS software:

HKEY_LOCAL_MACHINE\SOFTWARE\udcp

This registry key must contain the following entries:

- Driver: REG_SZ
This setting must contain the name of the driver executable including the full path, for example c:\program files\manufacturer\driver\udcp.exe. The POS application (server) will use this setting to actually start the driver and also to check if the driver has been launched successfully. This registry setting must be set on installation of the driver.

- Server: REG_SZ
This setting must contain the name of the server application i.e. the POS application including the full path. The driver must use this setting in order check if the server application is running. Only when the server is not running the driver is allowed to offer a more extended user interface. This registry setting must be set on installation of the POS software.

- CommandPath: REG_SZ
Both POS application and the dispenser driver use a designated directory to place the UDCP communication files. This registry setting must be set on installation of the driver.

- DispenserFamily: REG_ SZ
The dispenser family setting enables the POS software to identify the particular type of dispenser. In combination with Message Code (see 0. Message file) this setting uniquely identifies each dispenser driver's message. The POS software may use this identification for translation purposes. This setting must be set on installation of the driver.
A range of id's will be assigned to each dispenser manufacturer. One id identifies a range of similar dispensers from one manufacturer. This setting must be numerical.

The separate appendix “Dispenser Family Codes” holds the administration for the distribution of dispenser family codes among the various dispenser manufacturers.

- ServerUDCPversion: REG_ SZ
This is a comma separated list of protocol versions that the POS software supports (see 1.3). This setting must be set on installation of the POS software.

- ClientUDCPVersion: REG_ SZ
This is a comma separated list of protocol versions that the client supports (see 1.3). This setting must be numerical and must be set on installation of the driver software.

- ActiveVersion: REG_ SZ
The POS software calculates the highest protocol version that both Client and Server support and puts the result in this registry key. This will be the actual version of the UDCP version that the client and server will use for communication.

Note that although the type of these registry settings is always REG_SZ some of these setting must contain numerical values only.

1.3 Version mechanism

A UDCP protocol version is an ordinal number. This document describes protocol version 1. Future versions will be 2, 3 and so on. The client and server both write a comma separated string that holds the version they support to the registry. The server determines the highest version that both client and server support, that will be the actual version that will be used for communication. This actual version will be written back to the registry. The server application will not function when there is no version that is supported by client and server.

1.4 Files

UDCP files are ASCII files. UDCP provides for three types of files: Process, Status and Signal files.

1.4.1 Process Files

Process files may contain commands or information for the receiving application to process. There are several types of process files:

Command files - These files are sent by the server application to the dispenser driver. They instruct the dispenser to execute some command. There are a variety of commands, for example to perform a morning clean or to dispense an entire recipe. A command file to start a dispense action will typically hold the complete recipe. UDCP only allows for one command per file.

Result files – These files are sent by the dispenser driver to the server in order to return the outcome of a command. Each command file must ultimately be confirmed with a result file.

Abort files – The server application may send an abort file in order to interrupt and cancel the execution of a command file by the driver.

Message files – These files are sent by the dispenser driver. They provide means for the driver to request input from the server application.

Message-Reply files – With these files the server application responds to a message file.

1.4.2 Status Files

Busy file – The busy file is an empty file. Its existence indicates that the dispenser is busy executing a command sent by the server application and is not ready to receive a next command. When this file is absent the driver is ready to receive the next command file. The dispenser driver must only create this file in response to a command file sent by the server application. Upon completion of the command, after writing the result file, the driver removes the busy file.

1.4.3 Signal Files

All process files are accompanied by a signal file. The existence of a signal file will basically alert the receiving application about a new file to process. As it is the existence of the file that triggers the receiving application the contents of the file is not important. The files will therefore be empty.

1.5 Files naming convention

File	Filename	Signal File
Command	cmd.dat	cmd.sig
Busy	busy.flg	
Results	result.dat	result.sig
Abort	abort.dat	abort.sig
Message	msg.dat	msg.sig
Message-Reply	reply.dat	reply.sig

1.6 Mechanism

1.6.1 Signaling mechanism

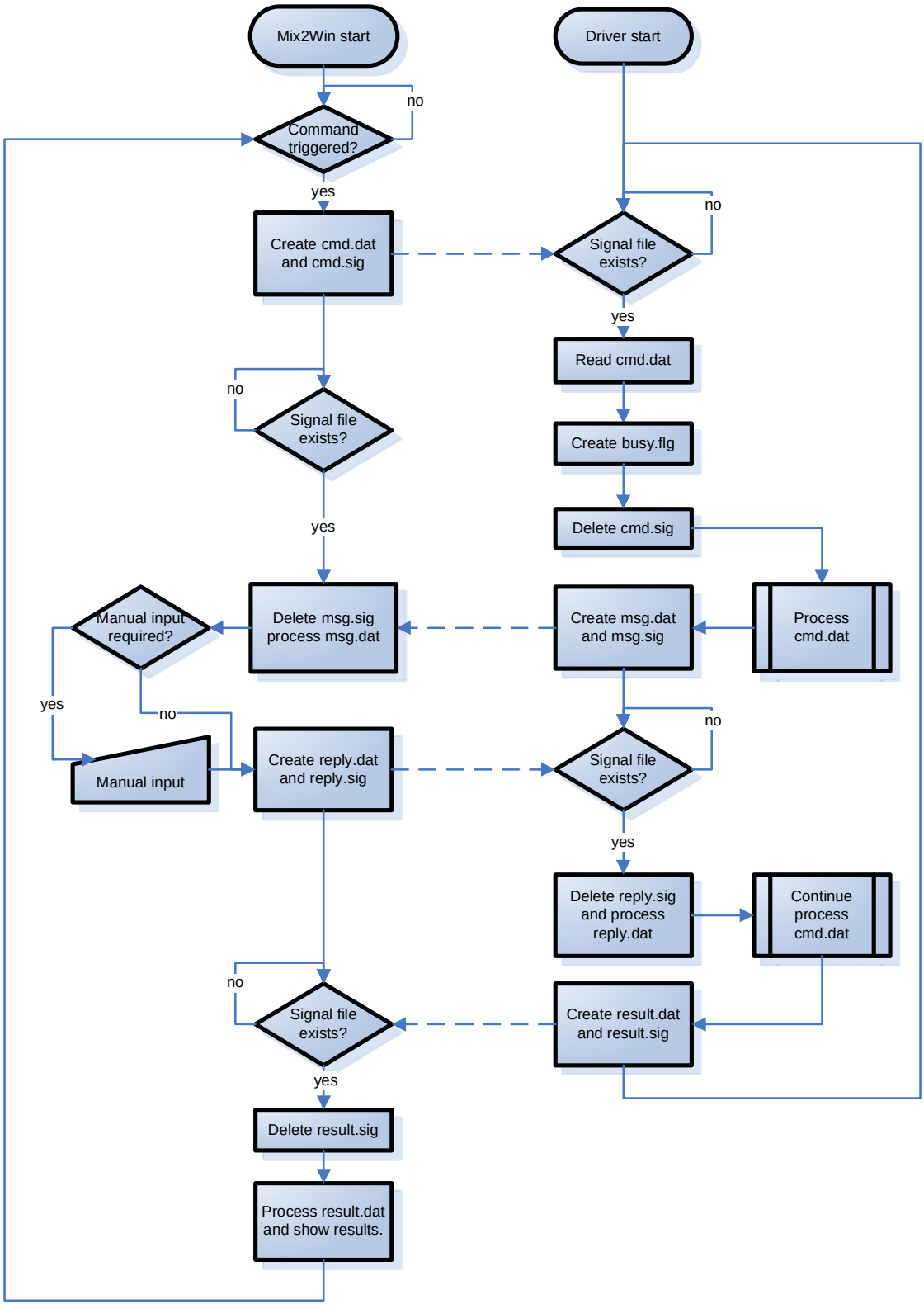
Both the server application and the dispenser driver implement a polling mechanism to look for new signal files. Whenever a new signal file is found the related process file must be processed. After processing the receiving application must remove the signal file. This will be the signal for the sending application that the related file has been processed.

1.6.2 Status mechanism

The server application also implements a polling mechanism to look for status files created by the dispenser driver. There is only one type of status file: the busy file. The server application will not send a new command file while the busy status file exists.

The general order of handling a command is described below:

1. The server application creates a cmd.dat command file and then a corresponding cmd.sig signal file.
2. The dispenser driver notices the cmd.sig signal file signal. It creates the busy.flg status file informing the server application it started processing the cmd.dat command file.
3. The driver reads the cmd.dat file and deletes the cmd.sig file.
4. The driver starts processing the cmd.dat file.
5. Optionally the driver creates a msg.dat and after that a corresponding msg.sig file. The msg.dat may for example contain an instruction to place the correct can.
6. When the server application notices the msg.sig signal file it processes the corresponding msg.dat file. After that it deletes the msg.sig file.
7. The server application responds to the driver in the form of a message-reply file reply.dat, accompanied by a corresponding message-reply signal file, reply.sig.
8. The driver notices the message-reply signal file. It processes the server application’s response and deletes the signal file.
9. Optionally the server application may send an abort.dat file with corresponding abort.sig file to have the driver cancel the execution of the current command.
10. When the driver finishes processing the command it informs the server application by creating a result.dat file and a corresponding result.sig file. The result.dat file holds the result code of the process which may be a success or an error code.
11. The driver deletes the busy.flg file informing the server application it’s ready to receive the next command.
12. The server application notices the result.sig signal file and processes the corresponding result.dat file. The server application then deletes the result.sig signal file.


Mix2Win start
Driver start
Command triggered?
no
yes
Create cmd.dat and cmd.sig
Signal file exists?
no
yes
Read cmd.dat
Signal file exists?
no
yes
Create busy.flg
Delete cmd.sig
Manual input required?
Delete msg.sig process msg.dat
Create msg.dat and msg.sig
Process cmd.dat
yes
no
Manual input
Create reply.dat and reply.sig
Signal file exists?
no
yes
Delete reply.sig and process reply.dat
Continue process cmd.dat
Signal file exists?
no
yes
Create result.dat and result.sig
Delete result.sig
Process result.dat and show results.

1.7 Driver Startup

When the server application launches the driver it will wait until the driver is ready. For this reason it creates a busy file which the driver has to remove when it's ready to receive commands from the server. The removal of this file will be the signal to the server application that the driver is ready. This is the only occasion where the busy file is generated by the server application.

COMMAND file

The command file is a standard ASCII file. The purpose of this file is to have the dispenser execute exactly one function, for example: one dispense, clean or canister refill action.

The command file contains a set of command keys. The general rules that apply to this file and the command keys are:

- Command keys are preceded by an “@” (ASCII code 64) and optionally followed by parameters.
- A command key is always contained on a single line.
- A space (ASCII code 32) is used to separate the command key from the parameters and between the parameters themselves.
- Parameters containing spaces are enclosed in quotes (ASCII code 34). The enclosing quotes are not part of the parameter value. Two consecutive quotes represent a single quote in the parameter value.
- The decimal separator in parameters representing a number is always a “.” (ASCII code 46)
- A command file always contains only a single command block. This block starts with @RUN and ends with @END
- Parameters can be optional. When an optional parameter is left out, but it has a succeeding parameter, then the left out optional parameter must be replaced by *UNK. (see example 1.14 Cantype key).

This document shows the command keys in `blue Courier font`. Additional comments are preceded by a double backslash // and must be left out of any files.

A command file is always accompanied by a command.sig signal file as prescribed by the signaling mechanism (see 1.6.1 Signaling mechanism). When the dispenser driver finishes interpreting the command file it must remove the command.sig file. To keep the server application from sending the next command file immediately it must create the busy.flg status file before it removes the command.sig file.

1.8 Command block

```
@RUN
...
@END
```

All command keys are grouped in a command block. This block starts with the @RUN key and ends with the @END key. Lines before the first start and after the first end key will be ignored. This implies that a command file can only contain a single command block. The number of command keys inside a command block is limited. It can only contain one function for the dispenser to execute. That means at most one dispense, clean or refill action.

1.9 Unit type key

```
@WGH <Unit type>
```

The @WGH key specifies the unit on which the amounts are based. This command is optional. When this command key omitted the default unit type (milliliter) is used.

Parameter <unit type> can have the following values:
0: Amounts are based on milliliter (default)
1: Amounts are based on grams

Examples:
```
@WHG 0          // to set unit to milliliters
```

1.10 Unit size key

```
@UNT <Metering unit> <Fraction>
```

The @UNT key specifies the exact unit size used for dispense amounts (See 1.15 Dispense Canister key/ 1.16 Dispense Colorants Key). The <metering unit> parameter is specified in milliliter or gram, based on the @WGH key. The <fraction> parameter specifies in how many fractions the metering unit is divided. The amount <metering unit> / <fraction> is called a shot, which is a typical notation for amounts in the paint industry.

Examples (when @WGH is 0):
```
@UNT 1.0 1.0        // unit size is 1.0 ml
@UNT 31.25 48       // unit size is 0.651 ml, which is a metric shot
@UNT 31.25 96       // unit size is 0.326 ml, which is half a metric shot
@UNT 29.30 48       // unit size is 0.610 ml, which is a US shot
```

Examples (when @WGH is 1):
```
@UNT 1.0 1.0        // unit size is 1.0 g
```

1.11 Lot quantity

```
@LQT <Number of cans to dispense>
```

The @LQT key specifies the number of cans to dispense for this formula. This parameter is optional. When omitted the default value for this key is 1.

Examples:
```
@LQT 1          // the formula is dispensed once, same as when this key is not present.
@LQT 99         // the formula is dispensed 99 times in 99 different cans.
```

1.12 Product key

```
@PRD <Product name>
```

The @PRD key specifies the product name for the formula. This parameter is optional. The dispenser may show this product name when the dispense process is busy.

Examples:
```
@PRD “Product ABC”
@PRD ProductABC
@PRD ABC
```

1.13 Base key

```
@BAS <Base name>
```

The @BAS key specifies the base paint name for the formula. This parameter is optional. With this parameter the dispenser is able to prompt the operator to place a can that contains the correct basepaint. Most cans are prefilled with base paint of a certain product, so the operator needs to select a can with the correct size, and product-base combination.

Examples:

```
@BAS “Base ABC”
@BAS BaseABC
@BAS ABC
```

1.14 Cantype key

```
@CAN <Cantype name> <Content [ml]> <Height [mm]>
```

The @CAN key specifies the name and size of the can in which to dispense the formula. This parameter is optional. The individual parameters are all optional as well. With this key the dispenser may prompt the operator to place the correct can. Most cans are prefilled with base paint of a certain product, so the operator needs to select a can with the correct size, and product-base combination.

The first parameter specifies the name, the second parameter the total contents of the can in ml, the third parameter is the height of the can in millimeters. The height parameter must be an integer value. Note that the content is always specified in ml and does not depend on the @WGH and @UNT keys.

Examples:

```
@CAN *UNK 4000 200              // Can has 4 Liter content and 20 cm height, name
                                // unknown
@CAN “Large bucket” 3000 200
@CAN BigBin 2000                // Can has 2 liter content, height unknown
```

Notes:

- The @SHL key (see 1.19 Shelf Handling) takes preference over the @CAN key. This means that when both a @SHL and @CAN occur within a command block then the Can Height parameter will be ignored.
- When there is no @SHL key and the dispenser is equipped with a automatic shelf the movement of that shelf will default to:
  1) No can height: preposition: move the shelf all the way down, position: move the shelf up until sensor blocks, postposition: move the shelf all the way down again.
  2) Can height available: preposition: move the shelf down to can height + margin, position: move until sensor blocks, postposition: lower to can height again + margin. Margin must be a driver setting.

1.15 Dispense Canister key

```
@CNX <Canister number 1> <Canister number 1 amount > … <Canister
number n> <Canister number n amount>
```

The @CNX key is a dispense command. The parameters are canister numbers paired with their corresponding amounts. The @CNX key may occur more than once within a command block even in combination with multiple @CNT keys (see 1.16 Dispense Colorants Key). All @CNX and @CNT keys together form the complete recipe.

The dispenser driver must interpret the sequence of @CNX/@CNT keys as a set of batch instructions to be handled in sequence. The dispenser driver is free to handle the sequence of dispense instructions that are contained in a single @CNX/@CNT key.

The @CNX key can only address canister numbers from 1 to the number of canisters on the dispenser. The amount is specified in the unit defined by the @WGH and @UNT keys (shots).

Examples:

```
@CNX 5 2.5 1 3.75  // canisters 5 amounts 2.5 and canister 1 3.75 shots, no sequence
                   // prescribed

@CNX 1 3.75        // first canister 1 3.75 shot and then canister 5 2.5 shots
@CNX 5 2.5
```

Notes:

- The @CNX/@CNT keys cannot be used in combination with a command extension key (See 1.21 Command extension key) in one command block.
- A dispense command may require the user to place a can. A @CAN key is therefore recommended.

1.16 Dispense Colorants Key

```
@CNT <Colorant name 1> <Colorant 1 amount > … <Colorant name n>
<Colorant n amount>
```

The @CNT key is a dispense command like @CNX (see 1.15 Dispense Canister key) based on colorants instead of canisters. The parameters of the @CNT key are colorant names paired with their corresponding amounts. The dispenser driver makes the translation from colorant name to canister number. This is not necessarily a 1 to 1 translation as there might be canisters that have the same colorant. In such a case the dispenser driver decides which canister to use.

The @CNT key relates to the @CNX key with regard to the fact that it also may appear several times within a command block even in combination to the @CNX key. Subsequent @CNT/@CNX keys must be handled in the sequence in which they appear in the command block. The sequence of dispensing colorants from one @CNT key is up to the driver.

If an unknown colorant name is specified, the formula will be refused by the driver. The colorant amount is specified in the unit defined by the @WGH and @UNT keys (shot).

Examples:

```
@CNT “AX” 2.5 B 2.45   // colorants AX 2.5 shots and B 2.45 shots, no sequence
                       // prescribed
@CNT AX 2.5            // first colorant AX 2.5 shots and then WTY 3 shots
@CNT WTY 3
```

Notes:

- The @CNX/@CNT keys cannot be used in combination with a command extension key (See 1.21 Command extension key) in one command block.
- A dispense command may require the user to place a can. A @CAN key is therefore recommended.

1.17 Canister Accuracy

```
@ACX <Canister number 1> <Canister number 1 accuracy > … <Canister
number n> <Canister number n accuracy>
```

The @ACX key sets the dispense tolerance for one or more canisters. The parameters are canister numbers paired with their corresponding accuracy. The accuracy is a percentage that applies to the amount parameter of the @CNT/@CNX key. That means that the dispenser must dispense from a canister within the limits <amount> - <amount> * <accuracy>/100 and <amount> + <amount> * <accuracy>/100

The dispenser driver uses the accuracy parameter to optimize dispensing for speed. When the accuracy is set to a higher level the dispense may choose to dispense faster and less accurate.

The @ACX/@ACT keys are optional. When absent the dispenser driver uses its default accuracy settings.

An @ACX key may appear several times in a command block even in combination with multiple @ACT keys (see 1.18 Colorant Accuracy). The @ACX key applies to all subsequent @CNT/CNX keys. The accuracy for a certain canister in a @CNX/@CNT key is taken from the last @ACX key in which it was mentioned. In other words @ACX/@ACT keys override previous ones.

An accuracy of 0 resets the accuracy setting for a canister to the default value.

Example:

```
@CNX 1 3.75          // dispense canister 1 3.75 shots, default accuracy
@ACX 1 2 5 10        // set accuracy canister 1 to 2 percent canister 5 to 10 percent
@CNX 1 2 5 2.5       // dispense canister 1 2 shots and 5 2.5 shots
@ACX 5 0             // set canister 5 accuracy to default
@CNX 5 6 1 3         // dispense canister 1 3 shots with accuracy still 2 percent and
                     // canister 5 6 shots, default accuracy
```

1.18 Colorant Accuracy

```
@ACT <Colorant name 1> < Colorant name 1 accuracy > … < Colorant
name n> < Colorant name n accuracy >
```

The @ACT sets the dispense tolerance for one or more colorants. This key is the colorant variant of the @ACX key (see 1.17 Canister Accuracy). The parameters of the @ACT are colorant names paired with their corresponding accuracy. As there might be multiple canisters that contain the same colorant a colorant name and accuracy may apply to several canisters. The accuracy is a percentage that applies to the amount parameter of the @CNT/@CNX key. That means that the dispenser must dispense from a canister within the limits <amount> - <amount> * <accuracy>/100 and <amount> + <amount> * <accuracy>/100

The dispenser driver uses the accuracy parameter to optimize dispensing for speed. When the accuracy is set to a higher level the dispense may choose to dispense faster and less accurate.

The @ACT/@ACX keys are optional. When absent the dispenser driver uses its default accuracy settings.

An @ACT key may appear several times in a command block even in combination with multiple @ACX keys (see 1.17 Canister Accuracy). The @ACT key applies to all

subsequent @CNT/CNX keys. The accuracy for a certain canister in a @CNX/@CNT key is taken from the last @ACT/@ACX key in which it was mentioned. In other words @ACT/@ACX keys override previous ones.

An accuracy of 0 resets the accuracy setting for a colorant to the default value.

Example:

```
@CNT “WTY” 3.75          // dispense colorant WTY 3.75 shots, default accuracy
@ACT WTY 2 WTZ 10        // set accuracy colorant WTY to 2 percent WTZ to 10 percent
@CNX WTY 2 “WTZ” 2.5     // dispense colorant WTY 2 shots and WTZ 2.5 shots
@ACX WTZ 0               // set colorant WTZ accuracy to default
@CNX WTZ 6 WTY 3         // dispense colorant WTY 3 shots with accuracy still 2
                         // percent and colorant WTZ 6 shots, default accuracy
```

1.19 Shelf Handling

```
@SHL <Preposition> <Position> <Postposition>
```

The shelf handling key specifies the height of an automatic shelf before, during and after dispensing. The parameters have the following meaning:

Preposition: Before asking the user to insert the can (to give enough room to insert the can)
Position: Before tinting (to check the correct size of the can inserted)
Postposition: After tinting (to remove the can)

Position and Postposition can have the following values:

> 0: Height of the shelf in mm (relative to the top position)
0: Move until sensor locks (alias all up)
-1: Move the shelf completely down
-2: Reference plane (a known intermediate equipment dependant position)
-3: No Movement

Preposition can have all above values except for -3: no movement

This key only applies to dispensers that are equipped with an automatic shelf and is optional.

Example:

```
@SHL -1 0 -1        // Moves the shelf all down before and after dispensing (this is, in
                    // any case the default behavior).

@SHL -2 0 -2        // Moves the shelf to the reference plane position before and after
                    // dispensing.

@SHL -1 400 -3      // Moves the shelf all down, move it down 40 cm and then keep it
                    // there after dispensing.
```

Notes:

- The @SHL key takes preference over the @CAN key. This means that when both @SHL and @CAN with a can height exist within one command block the height parameter from the CanType key (see 1.14 Cantype key) will be ignored.
- When there is no @SHL key and the dispenser is equipped with a automatic shelf the movement of that shelf will default behavior described in 1.14 Cantype key.

1.20 Punch

```
@PNC <Punch Action>
```

The punch key handles the punching of cans by the dispenser. The punch action parameter can have the following values:

0: Don't do punching.
1: Ask for confirmation before punching.
2: Punch directly without asking any confirmation.

Example:

```
@PNC 1        // Ask for user confirmation before punching the can.
```

Notes:

- The @PNC command is optional. When the @PNC command key is absent the driver decides how to handle punching if a punching device is present.

1.21 Command extension key

```
@EXT <parameter 1> … <parameter n>
```

The @EXT key allows extensions to the basic formula dispense commands.

1.21.1 RESET extension

```
@EXT RESET
@EXT RESET HARD
```

The RESET command instructs the driver to reset the dispenser. Standard reset only initializes the components of the dispenser that are uninitialized or are in an error state. RESET HARD initializes everything, just as if the power has been switched off and on. A command block can only contain one RESET command. If a command block contains a RESET command it cannot contain any other command.

The dispenser sets the result key in the general section of the result file.

Notes:

- The RESET commands do not affect the CanisterContents section in the result file.

Examples:

```
@RUN
@EXT RESET            // Reset, fast, fixes most error situations
@END

@RUN
@EXT RESET HARD       // Reset, time consuming, fixes virtually all error situations
@END
```

1.21.2 MACHINE_INFO extension

The MACHINE_INFO extensions provide means for the server application to obtain the dispensers machine configuration and available options. The following MACHINE_INFO extensions are available:

```
@EXT MACHINE_INFO FEATURES
@EXT MACHINE_INFO PUMPCOUNT
@EXT MACHINE_INFO CANISTERCOUNT
@EXT MACHINE_INFO CANISTERSIZE
```

The MACHINE_INFO FEATURES command enables the server application to get the features that dispenser supports. It also returns the driver's error log. The driver must put the following information in the result file:

```
[General]
Result = 0                      // signals that the output key is valid
Output = <Features code>

[Features]
Text1 = <FeatureText1>
Text2 = <FeatureText2>
…

[Errorlog]
Error1 = <ErrorText1>
Error2 = <ErrorText2>
…
```

Notes:

- Features Code is a bit field which encodes the dispenser's specific features. Each bit represents a feature. The exact meaning of each bit field depends on the dispenser family.
- The Features section contains human readable texts that describe the features of the dispenser. The maximum number of Texts is 100.
- The ErrorLog Section contains the last 50 errors in human readable format. The errors are numbered chronologically, Error50 is the last error.

The MACHINE_INFO PUMPCOUNT and CANISTERCOUNT commands return the number of pumps and the number of canisters of the dispenser respectively in the output key of the result file.

```
[General]
Result = 0                      // signals that the output key is valid
Output = <count>
```

Examples:

```
@RUN
@EXT MACHINE_INFO FEATURES      // Get dispenser features
@END

@RUN
@EXT MACHINE_INFO PUMPCOUNT     // Get number of pumps
@END
```

The MACHINE_INFO CANISTERSIZE command returns the configuration and content of all canisters in the CanisterContents section of the Result file.

```
[CanisterContents]
Canister_1_ml = <contents of canister 1 in ml>
Canister_1_size_ml = <size of canister 1 in ml>
Canister_1_warn_ml = <warning level of canister 1 in ml>
Canister_1_min_ml = <minimum level of canister 1 in ml>
Canister_1_dosed_ul = <dispensed amount from canister 1 in ul>
Canister_1_ColorantName = <name of the colorant in canister 1>

Canister_<N>_ml = <contents of canister <N> in ml>
Canister_<N>_size_ml = <size of canister <N> in ml>
Canister_<N>_warn_ml = <warning level of canister <N> in ml>
Canister_<N>_min_ml = <minimum level of canister <N> in ml>
Canister_<N>_dosed_ul = <dispensed amount from canister <N> in ul>
Canister_<N>_ColorantName = <name of the colorant in canister <N>>
```

<N> is the number of canisters the dispenser has. When a canister is not in use the Canister_<x>_ml content key reports -1.

Notes:

- The MACHINE_INFO commands are for information purposes only; they do not change the CanisterContents section in the results file.
- The MACHINE_INFO commands must return Canister_<x>_dosed_ul parameters. These parameters must be set to 0.
- A MACHINE_INFO command cannot be combined with any other commands in one command block.

1.21.3 VERSION extension

```
@EXT VERSION
```

The version command enables the server application to retrieve the version of the dispenser driver.

The dispenser driver returns the version number in the output key in the result file (see also 0.RESULTS file). The dispenser driver also returns version information in human readable form in the version section:

```
[General]
Result = 0                  // signals valid result
Output = <version>

[Version]
Text1 = <version information text1>
Text2 = <version information text2>
..
```

Version is multiplied by 10000 as an integer number. E.g. version 1.00.00 or 1.00 are both represented by 10000. The maximum number of lines in the version section is 100.

Notes:

- The VERSION command is for information purposes only. The command does not change the CanisterContents section in the results file.
- A version command must not be combined with other commands in one command block.

Examples:

```
[General]
Result = 0                  // signals that the OUTPUT key is valid
Output = 20000              // returns version 2.0000

[Version]
Text1 = 2.0000 Beta
Text2 = Version for test purposes only
```

1.21.4 PURGE extension

```
@EXT PURGE <canister>
```

The PURGE command enables the server application to instruct the dispenser to purge one or all canisters. The purge amount is a dispenser or driver setting.

Parameter <canister> can have the following values:
0: Purge ALL canisters
1...CanisterCount : Purge the specified canister. Get number of canisters by using @EXT MACHINE_INFO CANISTERCOUNT (see 1.21.2).

Examples:

```
@EXT PURGE 0        // Purge all canisters
@EXT PURGE 2        // Purge canister 2
```

Notes:

- The dispenser driver may use a smart approach in purging the canister(s). This means the amount of colorant used for purging may depend on the time that the canister was not in use.
- The CanisterContents including the Canister_<x>_Dosed_ul key (see 1.23 CanisterContents section) in the results file must be updated accurately to reflect the amount of colorant that was used for purging.
- A PURGE command may be aborted by the server application using an abort file.
- A PURGE command cannot be combined with another command within the same command block.

1.21.5 PURGECOL extension

```
@EXT PURGECOL <Colorant Name>
```

The PURGECOL command enables the server application to instruct the dispenser to purge a certain colorant. The PURGECOL command applies to all canisters that contain the colorant <Colorant Name>.

The parameter <Colorant Name> is the name of the colorant to be purged. The PURGECOL command cannot be used to purge all canisters at once.

Notes:

- The dispenser driver may use a smart approach in purging the canister(s). This means the amount of colorant used for purging may depend on the time that the canister was not in use.
- The CanisterContents including the Canister_<x>_Dosed_ul key (see 1.23 CanisterContents section in the results file must be updated accurately to reflect the amount of colorant that was used for purging.
- A PURGECOL command may be aborted by the server application using an abort file.
- A PURGECOL command cannot be combined with another command within the same command block.

Examples:
```
@EXT PURGECOL “WTX”        // Purge colorant WTX

@EXT PURGECOL “1”          // Purge colorant “1”
```

1.21.6 CLEAN extension

```
@EXT CLEAN <canister>
```

The CLEAN command allows the server application to instruct the dispenser to clean one or all canisters. The dispenser may use a brush or jet-cleaning or another method to clean the pump(s) of a canister. In any case in cleaning no colorant must be dispensed.

Parameter <canister> can have the following values:
0 : Clean all canisters.
1..CanisterCount : Clean the specified canister. Get number of canisters by using Machine_info CanisterCount (see 1.21.2).

Examples:
```
@EXT CLEAN 0 // Clean all canisters

@EXT CLEAN 2 // Clean canister 2
```

Notes:
- The CanisterContents section is not changed by a CLEAN command as there are no colorants used in cleaning. The CanisterContents section must however be returned completely including the Canister_<x>_Dosed_ul key.
- A CLEAN command may be aborted by the server application using an abort file.
- A CLEAN command cannot be combined with another command within the same command block.

1.21.7 CLEANCOL extension

```
@EXT CLEANCOL <Colorant Name>
```

The CLEANCOL command enables the server application to instruct the dispenser to clean canisters which contain colorant <Colorant Name>. The dispenser may use a brush or jet-cleaning or another method to clean the pump(s) of a canister. In any case in cleaning no colorant must be dispensed.

The parameter <Colorant Name> is the name of the colorant to be purged. The CLEANCOL command cannot be used to clean all canisters at once.

Examples:

```
@EXT CLEANCOL “WTX” // Clean all canisters containing WTX
```

Notes:
- The CanisterContents section is not changed by a CLEANCOL command as there are no colorants used in cleaning. The CanisterContents section must however be returned completely including the Canister_<x>_Dosed_ul key.
- A CLEANCOL command may be aborted by the server application using an abort file.
- A CLEANCOL command cannot be combined with another command within the same command block.

1.21.8 FILLCANISTER extension

The FILLCANISTER extensions provide means to the server application to handle the refilling of canisters. The following FILLCANISTER extensions are available:

```
@EXT FILLCANISTER PREPARE <canister>
@EXT FILLCANISTER ADD <canister> <amount in ml>
@EXT FILLCANISTER SET <canister> <amount in ml>
```

The first step in this process typically is to call the fill canister prepare command which causes the dispenser to move the specified canister to the filling position. After filling the specified canister the server application must send the added amount, or set amount to the driver. The driver uses this information to update its administration of canister contents.

Parameter <canister> must have a value between 1 and CanisterCount, where CanisterCount can be retrieved using MACHINE_INFO CANISTERCOUNT (see 1.21.2).

The FILLCANISTER ADD Parameter <amount in ml> is an integer number, specifying the amount in ml to be added to the canister contents. The following values are possible:

Positive value: The specified amount is added to the current canister content. If canister content would become higher than the canister size than the canister content is set to the canister size. The canister size can be obtained using the MACHINE_INFO CANISTERSIZE command (see 1.21.2).
Negative value: The specified amount is subtracted from the current canister contents. When the content becomes less than 0, the content is set to 0. E.g. if the current content is 1000 ml, a value of -400 would set the contents to 600 ml and -9000 would set the contents to 0.
0: The canister content is not changed.

Analogue to the FILLCANISTER ADD command, there is also a FILLCANISTER SET command. This command sets the canister content to the specified content in ml. This amount cannot be a negative value. When the amount is set to a value higher than the canister size than the canister amount is set to the canister size.

Examples:

```
@EXT FILLCANISTER PREPARE 2     // (Position canister 2 on the filling position)

@EXT FILLCANISTER ADD 2 1000    // (Add one liter to canister 2)

@EXT FILLCANISTER ADD 2 0       // (Only get canister contents of all canisters)

@EXT FILLCANISTER SET 2 999999          // (Sets canister 2 to the top…)

@EXT FILLCANISTER ADD 2 -999999         // (Empty canister 2…)
```

Notes:

- The PREPARE and ADD/SET commands must be sent in separate command blocks.
- A FILLCANISTER command cannot be combined with another command within one command block.
- A FILLCANISTER PREPARE command may be aborted by the server application.
- FILLCANISTER ADD/SET commands are for driver administration purposes only. They do not cause a dispenser action and cannot be aborted. The CanisterContents section of the results file must be updated with the information from the FILLCANISTER ADD/SET command.

- A FILLCANISTER ADD/SET command will be ignored when the specified canister is not in use.
- Typically the driver checks the contents for every canister before and after dispensing. A warning is given when the canister contents falls below the Warning Level after dispensing. Dispensing is not possible when the canister contents would fall below the Minimum Level after dispensing.

1.21.9 SETCANCOLNAME extension

```
@EXT SETCANCOLNAME <Canister> <Colorant Name>
```

This key enables the server application to assign a colorant name to a canister number. This way the server application can synchronize the driver's canister administration with its own.

Parameter <canister> is mandatory and must be between 1 and CanisterCount. Get number of canisters by using @EXT MACHINE_INFO CANISTERCOUNT (see 1.21.2).

The <Colorant Name> parameter is also mandatory.

Examples:

```
@EXT SETCANCOLNAME 2 WTY          // Canister 2 contains colorant WTY
```

Notes:

- The SETCANCOLNAME command is for administration purposes only; it does not change the CanisterContents section in the results file.
- A SETCANCOLNAME command cannot be combined with any dispense, purge or clean command in one command block.

1.21.10 SHOWSERVICEWINDOW extension

```
@EXT SHOWSERVICEWINDOW <password>
```

This key enables the server application to open the dispenser's maintenance service window. This service window is the only time the driver is allowed to interface with the user.

Typically a command file with this command is generated when the user selects a "Service window" menu item or button in the server application. The parameter <password> is optional. The driver may use the password to determine the rights the user has in the service window.

The implementation of the service window itself is driver specific. The main purpose is to give access to dispenser specific configuration settings.

Examples:

```
@EXT SHOWSERVICEWINDOW

@EXT SHOWSERVICEWINDOW 1002  // show the service window with 1002 as password
```

Notes:

- The SHOWSERVICEWINDOW key must be placed as the only command key between the start and end key in a command file.
- The CanisterContents section in the RESULTS file may be changed as a result of this command.
- The busy status file must be created when the service window is opened. It must only be removed when the service window is closed.

BUSY file

The busy status file (busy.flg) is an empty file that indicates the dispenser is busy handling a command. The dispenser driver is responsible for the creation and deletion of this file. The only exception to this is when the server application launches the driver it will create a busy file and wait until the driver removes it.

This file is created when the driver receives a new command file from the server application. It is only deleted when the dispenser completely finishes processing the command file, successfully or with an error report. This allows the server application to synchronize commands with the dispenser driver; while a busy file exists the driver is still busy processing a command; when there is no busy status file the driver is ready to receive a new command.

Notes:

- The busy file has no contents; its mere existence indicates that the dispenser is still busy processing a previous command file.

RESULTS file

The result file (result.dat) reports back the result of a command, the status of the dispenser, the content of its canisters and the error log. The server application may parse this file in order to report this back to the user and to update it's administration. The dispenser driver creates this file when it completes the execution of a command file. After creation of this file the busy status file must be removed by the dispenser driver which informs the server application the driver is ready to receive a new command file.

Notes:

- The result file is accompanied by a result.sig signal file as prescribed by the signaling mechanism (see 1.6.1 Signaling mechanism).

The results file (result.dat) is an ASCII file, formatted as a standard windows ini file. The following sections are mandatory:
- General
- CanisterContents

Optionally it may also contain the following sections:
- Features: Only MACHINE_INFO FEATURES returns this section (See 1.21.2 MACHINE_INFO extension)
- ErrorLog: Only MACHINE_INFO FEATURES returns this section (See 1.21.2 MACHINE_INFO extension)
- Version: Only VERSION returns this section (See 1.21.3 VERSION extension)

1.22 General section

```
[General]
Result = <result code>
Output = <integer value>
```

The results file must contain a general section. Within that section there are two key words: Result and Output. All commands return a result code as an integer value in the Result key, so all results files contain a Result key. The result code is 0 (E_OK) or another value indicating an error. Only some EXT commands return their output in the Output key. Only for these commands the results file will contain an Output key.

The result codes returned in the Result key are grouped together as follows:
0 (E_OK): The command file's execution completed without errors.
1 – 31: Reserved in protocol for errors related to command file parsing
>=32: Driver specific error codes.

The errors related to command file parsing:
1 (E_UNKNOWN_COMMAND): Unknown command: the command file contains an unknown key.
2 (E_INVALID_NR_OF_PARAMETERS): The command file contains a key with too few mandatory parameters (note: client should ignore any extra undefined parameters).
3 (E_COMMAND_MISSING): The command file does not contain any command key to execute. A command file without any @CNT, @CNX or @EXT command produces this error.
4 (E_PARAM_VALUE_RANGE): The command file contains a key with an invalid parameter value.
5 (E_ABORTED): The command was aborted during execution.

The Output key is only used for the following extended commands (@EXT); see paragraph 1.21 for further information:

```
@EXT MACHINE_INFO FEATURES
@EXT MACHINE_INFO PUMPCOUNT
@EXT MACHINE_INFO CANISTERCOUNT
@EXT VERSION
```

1.23 CanisterContents section

The CanisterContents section is a mandatory section of the results file. It contains the configuration and contents of all canisters in the dispenser.

All parameters in the CanisterContents section are mandatory for each command:

```
[CanisterContents]
Canister_1_ml = <contents of canister 1 in ml>
Canister_1_size_ml = <size of canister 1 in ml>
Canister_1_warn_ml = <warning level of canister 1 in ml>
Canister_1_min_ml = < minimum level of canister 1 in ml>
Canister_1_ColorantName = <name of the colorant in canister 1>
Canister_1_dosed_ul = <dispensed amount from canister 1 in ul>

…
Canister_<N>_ml = <contents of canister <N> in ml>
Canister_<N>_size_ml = <size of canister <N> in ml>
Canister_<N>_warn_ml = <warning level of canister <N> in ml>
Canister_<N>_min_ml = <minimum level of canister <N> in ml>
Canister_<N>_ColorantName = <name of the colorant in canister <N>>
Canister_<N>_dosed_ul = <dispensed amount from canister <N> in ul>
```

<N> is the number of canisters the dispenser has.

<contents of canister x in ml> can have the following values (with x > 0 and x <= N):
>= 0: Amount of colorant in canister x in milliliters
-1: Canister is not in use

Notes:

- For every canister in the dispenser, the contents after the command has finished is specified in milliliters (ml) as an integer value. The first canister is “1”. All canisters must have consecutive numbers, so e.g. 1, 2, 3 is possible, but not 1, 3, 4 is not. The last canister N is also the number that can be retrieved using MACHINE_INFO CANISTERCOUNT (see 1.21.2).
- Canister_<x>_dosed_ul is the amount that was dispensed in ul (microliters). For commands that do not actually change the contents of the canisters (for instance MACHINE_INFO CANISTERCOUNT (see 1.21.2)) the Canister_<x>_dosed parameters must be 0.
- In case one or more canisters are equipped with a sensor, the content before dispensing minus the amount from the recipe might in extraordinary circumstances not match the content after dispensing. In all cases the Canister_x_dosed_u parameter reflects the correct dispensed amount.

ABORT file

The abort file is an empty file created by the server application. Its purpose is to abort the execution of the command file which the dispenser is currently processing. The dispenser driver polls for the existence of this file while executing a command file. When it detects this file it will abort the process as soon as possible.

The abort file complies with the signaling mechanism (see 1.6.1 Signaling mechanism). To inform the server application that it received the abort file the dispenser driver deletes the abort.sig file. After the current command has been successfully aborted it must clear the busy.flg status file to inform the server application it's ready to receive a next command file.

When a command is successfully aborted the dispenser driver sets the result key in the result file to 5 (E_ABORTED). The CanisterContents section is updated according to the amount that was dispensed or purged prior to the abortion of the command.

When the dispenser driver finishes the execution of the command normally before processing the abort.flg and abort.dat file then the dispenser driver ignores the abort attempt. It still deletes the abort.sig file in order to comply with the signaling mechanism.

There are three commands that can be aborted:

- Dispense commands (@CNT/@CNT)
- Purge commands (@PURGE/ @PURGECOL)
- Clean commands (@CLEAN/@CLEANCOL)

Message file

The dispenser uses the message file when it requires input from the server application. It's commonly used when processing commands or in error situations. The format of the file is based on user dialogs on the screen; however it's up to the server application how to process it. For instance, when the driver creates a message file informing the server application there is an error situation, the server application may show the error on the screen but depending on the error the server application might also automatically send a reset command.

The message file (msg.dat) is an ASCII file formatted as a standard Windows ini file. It contains a two mandatory sections Msg and MsgValues containing the following keys:

The Msg Section contains the following keys, are all mandatory:

```
[Msg]
Code = <Message Id>            // id of the message
DlgType = <Dialog Type>        // integer value specifying the message type
Buttons = <Button Bit Field>   // buttons to show on the message
Msg = <Message Text>           // message text
```

The Message Id uniquely identifies the message within a dispenser family. (See 1.2 Settings DispenserFamily). The server application can use this id for translation purposes.

The Dialog Type is an integer value that specifies the type of the message. It's typically represented by an icon. It can have the following values:

-1= Cancel previous message
0 = Unspecified (no icon)
1 = Warning (!)
2 = Error (stop)
3 = Information (i)
4 = Confirmation (?)

When the Dialog Type is -1 the driver requests the server application to close the previously opened message window. This is used to close messages that require no user action (i.e. progress messages) or when the dispenser detects that no user action is needed anymore (e.g. to close the "Place can" message when the dispenser detects a can in front of the can sensor).

The buttons parameter is an integer value specifying which buttons to display in the message window as a bit mask.

Button codes are:

1 = Yes
2 = No
4 = OK
8 = Cancel
16 = Abort
32 = Retry
64 = Ignore
128 = All
256 = No to All
512 = Yes to All
1024 = Help

Note:

- When the Button Bit Field is 0, then no buttons are displayed and no action is required from the user. The driver is required to send a cancel previous message (DlgType = -1) in order to remove such a message from the screen.

The Message Text contains the default text to be shown in the message window. The server application may hold a number of text translations and choose to show them instead. However when such a translation is not available it can use this default text.

The Message Text may hold the following placeholders to be replaced by parameters from the MsgValues section and placeholders to be replaced by line-breaks:

<brk>: to be replaced with by line-break.
x: to be replaced by parameter x from the MsgValues section.

The MsgValues holds the following parameters:

```
[MsgValues]
Count = N                          // Number of parameters in the MsgValues section
Parm1 = <Value Parameter 1>
…
ParmN = <Value Parameter N>
```

Count is the number of parameters in the MsgValues section. The Parameters keys must be in sequence from Parm1 to ParmN where N is the number of parameters.

The parameters must only be used for variable texts and must not contain placeholders (1 and <brk>). Parameters cannot be empty.

The MsgValues section is mandatory. When there are no parameters, the count key must be set to 0.

Example:

```
[Msg]
Code = 10
DlgType = 1                        // warning message
Buttons = 4                        // show OK button
Msg = Please fill canister $1$ with <brk> colorant $2$ and press
OK.

[MsgValues]
Count = 2
Parm1 = 3
Parm2 = WTY
```

The text from the example would become:

```
Please fill canister 3 with
colorant WTY and press OK.
```

Notes:

- The dispenser driver may request the server application to display a message 1 or more times before it requests the server application to close the message window. When a new message is requested before the previous message is closed, the server application will “overwrite” the previous message with the new one.
- When the message window is closed, it will always be removed from the screen completely, regardless of how many messages were requested. When a message is closed previous messages will not be shown. This is typically used for progress messages, like “Dispensing component A” followed by “Dispensing component B”, followed by a close request.
- The message file mechanism complies with the signal mechanism. This means that the message file msg.dat is accompanied by a signal file msg.sig. The server application must continuously poll for this signal file. By deleting the signal file the server application acknowledges to the dispenser driver that it has received the msg.dat file.

Message Reply file

The server application must use the message reply file reply.dat to reply to a message file from the dispenser driver. The message reply file is an ASCII file formatted as a standard windows ini file.

It contains a single section [Reply] containing the following key:

```
[Reply]
Button = <Button Number>
```

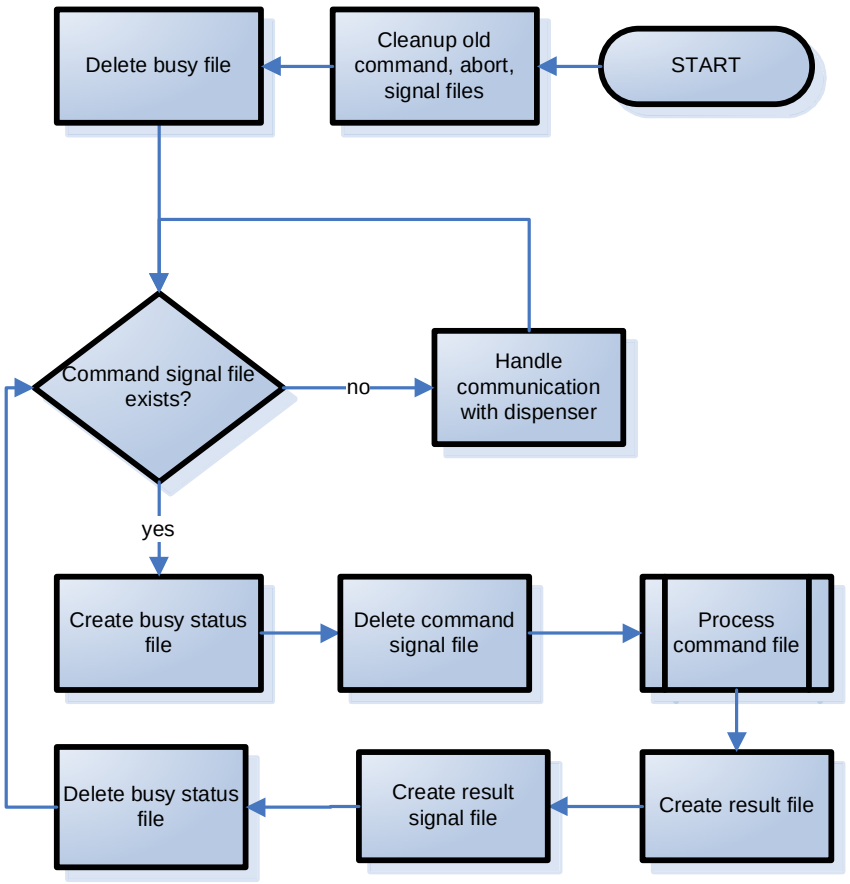
The button parameter is an integer value specifying the button pressed to close the message window. This can be any value from the buttons bitmap list (see chapter 0. Message file). E.g. 4 signals OK, but 3 is not a valid button code.

The message reply file reply.dat complies with the signaling mechanism. It's therefore accompanied by a reply.sig signal file.

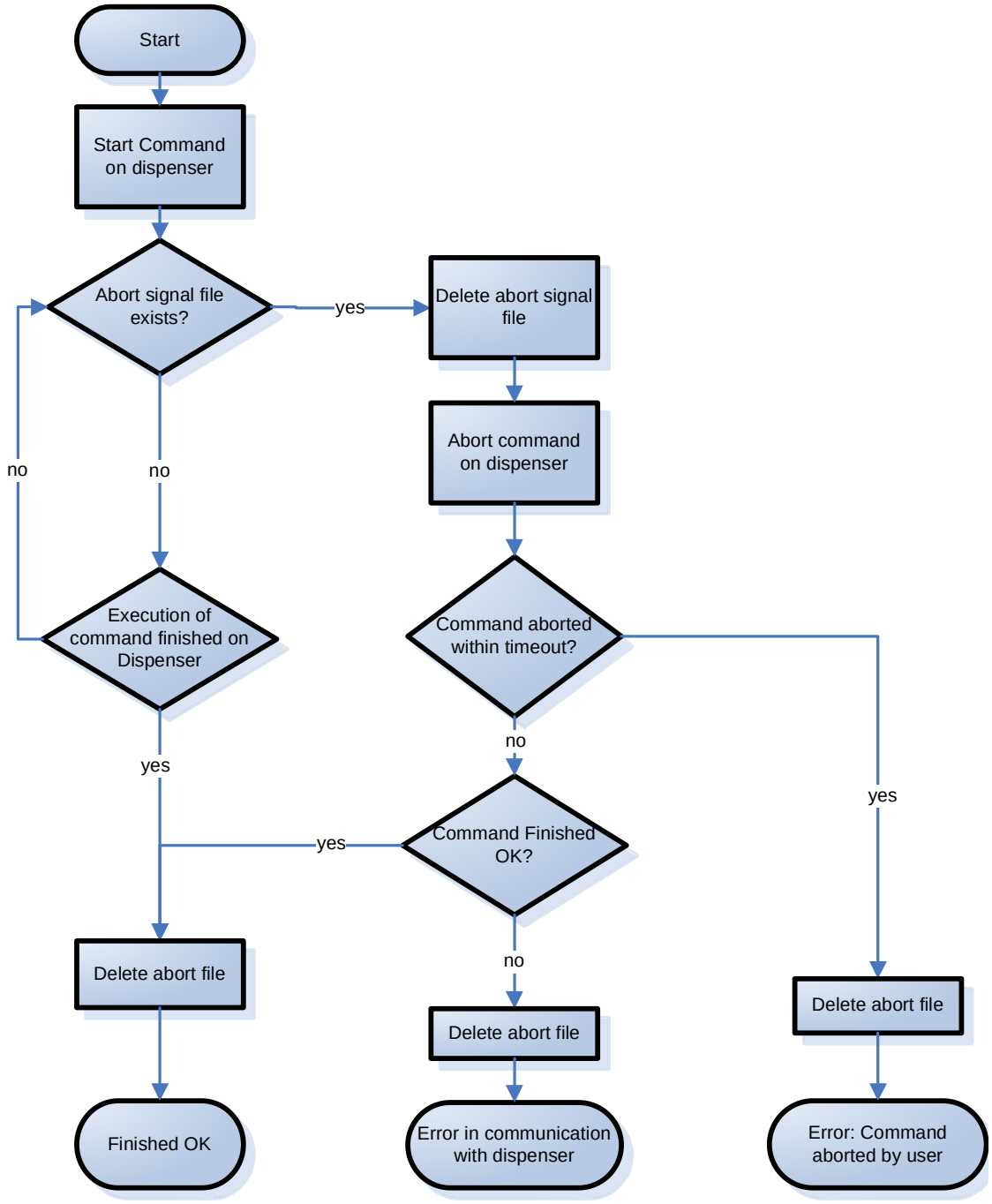
Note that not all message files must be answered by a message reply file.

Appendix A – Process flow

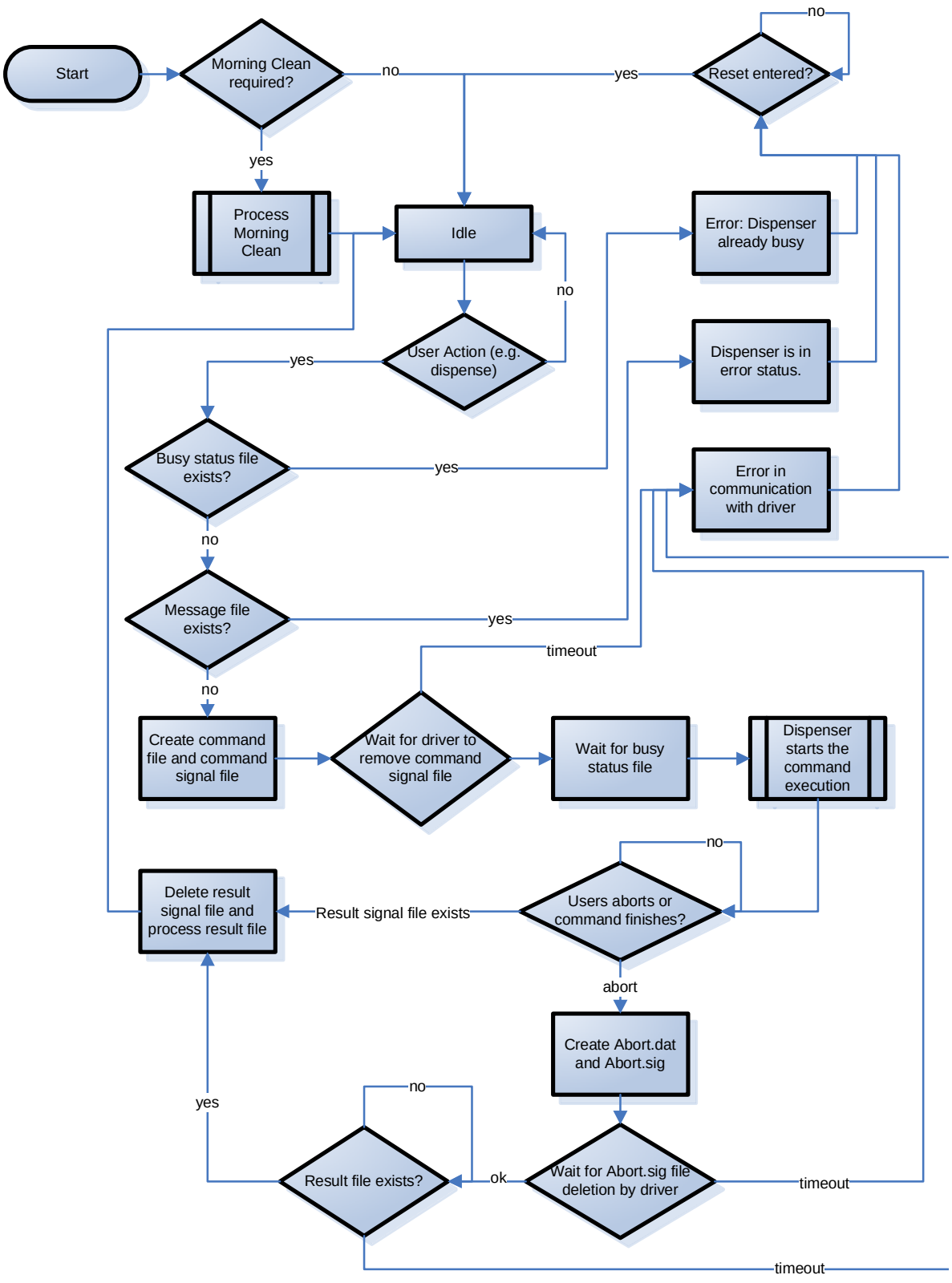
1.24 Device driver



1.25 Command process



1.26 Server application



Appendix B - Mix2Win UDCP usage

This chapter discusses AkzoNobel's Mix2Win usage of UDCP.

Current Mix2Win does not support dispensing multiple cans at once. The lot quantity (see 1.11) will only be used with a quantity of 1.

Mix2Win only support milliliters as unit type. The Unit type key (see 1.9) will therefore always be 0.

Mix2Win keeps its own canister administration. In communication with the dispenser driver it will therefore only address canisters instead of colorants.

Mix2Win will not use accuracy commands.

Appendix C - Definitions, acronyms, and abbreviations

Acronym / Word	Definition
UDCP	Universal dispenser communication protocol
Server	Front software application that issues commands to the dispenser/driver.
Client/Driver	The application that receives the commands and controls the dispenser hardware.
Dispenser	Hardware that actually dispenses the paint.
ASCII	American Standard Code for Information Interchange – character encoding for text based documents
Registry	Windows configuration settings repository
Shelf	Plateau on the dispenser hardware where to place the can to dispense
POS	Point of sales
Mix2Win	AkzoNobel's point of sales software

Appendix D - Examples files

This appendix contains examples files that were created by the VB6 POC software.

1.27 Result file

```
[General]
OUTPUT=12345678
RESULT=0
 [Features]
Text1=Type: Simulated dispenser
Text2=Contains 32 Canisters
Text3=Audio enabled for vivid expirience
[CanisterContents]
canister_1_ml=12790
canister_1_size_ml=20000
canister_1_min_ml=1000
canister_1_warn_ml=2000
canister_1_dosed_ul=0
canister_1_colorantName=A1
canister_2_ml=5565
canister_2_size_ml=10000
canister_2_min_ml=500
canister_2_warn_ml=1000
canister_2_dosed_ul=0
canister_2_colorantName=A2
canister_3_ml=2000
canister_3_size_ml=20000
canister_3_min_ml=1000
canister_3_warn_ml=2000
canister_3_dosed_ul=0
canister_3_colorantName=A3
canister_4_ml=5565
canister_4_size_ml=10000
canister_4_min_ml=500
canister_4_warn_ml=1000
canister_4_dosed_ul=0
canister_4_colorantName=A4
canister_5_ml=12790
canister_5_size_ml=20000
canister_5_min_ml=1000
canister_5_warn_ml=2000
canister_5_dosed_ul=0
canister_5_colorantName=A5
canister_6_ml=5565
canister_6_size_ml=10000
canister_6_min_ml=500
canister_6_warn_ml=1000
canister_6_dosed_ul=0
canister_6_colorantName=A6
canister_7_ml=12790
canister_7_size_ml=20000
canister_7_min_ml=1000
canister_7_warn_ml=2000
canister_7_dosed_ul=0
canister_7_colorantName=A7
canister_8_ml=5565
canister_8_size_ml=10000
canister_8_min_ml=500
canister_8_warn_ml=1000
canister_8_dosed_ul=0
canister_8_colorantName=A8
canister_9_ml=12790
canister_9_size_ml=20000
canister_9_min_ml=1000
canister_9_warn_ml=2000
canister_9_dosed_ul=0
canister_9_colorantName=A9
canister_10_ml=5565
```

canister_10_size_ml=10000
canister_10_min_ml=500
canister_10_warn_ml=1000
canister_10_dosed_ul=0
canister_10_colorantName=A10
canister_11_ml=12790
canister_11_size_ml=20000
canister_11_min_ml=1000
canister_11_warn_ml=2000
canister_11_dosed_ul=0
canister_11_colorantName=A11
canister_12_ml=5565
canister_12_size_ml=10000
canister_12_min_ml=500
canister_12_warn_ml=1000
canister_12_dosed_ul=0
canister_12_colorantName=A12
canister_13_ml=12790
canister_13_size_ml=20000
canister_13_min_ml=1000
canister_13_warn_ml=2000
canister_13_dosed_ul=0
canister_13_colorantName=A13
canister_14_ml=5565
canister_14_size_ml=10000
canister_14_min_ml=500
canister_14_warn_ml=1000
canister_14_dosed_ul=0
canister_14_colorantName=A14
canister_15_ml=12790
canister_15_size_ml=20000
canister_15_min_ml=1000
canister_15_warn_ml=2000
canister_15_dosed_ul=0
canister_15_colorantName=A15
canister_16_ml=5565
canister_16_size_ml=10000
canister_16_min_ml=500
canister_16_warn_ml=1000
canister_16_dosed_ul=0
canister_16_colorantName=A16
[ErrorLog]
Error1=6/9/2008 1:01:55 PM > EXT command accepted
Error2=6/9/2008 1:01:55 PM > MACHINE_INFO command found
Error3=6/9/2008 1:01:55 PM > Processing command.
Error4=6/9/2008 1:01:55 PM > </Received command:>
Error5=6/9/2008 1:01:55 PM > @END
Error6=6/9/2008 1:01:55 PM > @EXT MACHINE_INFO FEATURES
Error7=6/9/2008 1:01:55 PM > @RUN
Error8=6/9/2008 1:01:55 PM > <Received command:>
Error9=6/9/2008 1:01:55 PM > Command signal found.
Error10=6/9/2008 1:01:42 PM > Client Idle.

1.28 Command Files

Dispense command:

```
@RUN
@WGH 0
@UNT 29.753 48
@LQT 1
@PRD "Cetol Blur"
@BAS "M00"
@CAN "1L DustBin" 1000 *UNK
@CNX 3 12 8 44 31 4
@END
```

Return Features:

```
@RUN
@EXT MACHINE_INFO FEATURES
@END
```

Purge Command:

```
@RUN
@EXT PURGE 6
@END
```

1.29 Message Files

Place can message:

```
[Msg]
Msg=Place can $1$<brk>of size $2$
Dlgtype=3
buttons=12
Code=1
[MsgValues]
Count=2
Parm1=1L DustBin
Parm2=1 L
```

Cancel message:

```
[Msg]
Msg=
Dlgtype=-1
buttons=0
Code=0
[MsgValues]
Count=0
```

1.30 Message Reply Files

Reply ok button:

```
[Reply]
Button=4
```