Summa S Class 3

Programmer's Guide

Revision: 1123/1

Revision History

JUN	2023	Initial version.
AUG	2023	First release.
SEP	2023	Added eng_sdk@summa.com as general contact address.
NOV	2023	Added Cut-off with End Of Plot Command for DM/PL.

Notice

Please address all questions, comments or suggestions concerning this and other Summa manuals to:

Rochesterlaan 6
B-8470 Gistel
Belgium

Table of Contents

1 Introduction

The information contained in this package is designed to allow technically competent personnel to develop software and drivers compatible with the cutters from Summa. For any questions on this document, please contact **eng_sdk@summa.com**.

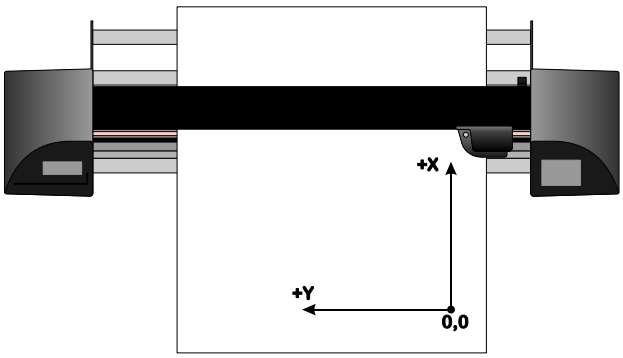
2 Cutter Commands

The cutter understands 3 languages: DM/PL, HP-GL and HP-GL/2. Because HP-GL and HP-GL/2 are almost identical, no difference is made in this document between both languages. Only a small subset of these languages is described and supported by the cutters from Summa.

The cutter also has a specific language for changing the parameters of the cutters such as the velocity and pressure. This language is called encapsulated language.

The **X-axis** is the direction in which the **media** moves, the maximum size of media in X-direction is 50 meter. It's advised to minimize the media movement by cutting shapes from the front to the back.
The **Y-axis** is the direction in which the **cut-head** moves, the maximum size varies depending on the model.

Illustration 2.1 - Media Orientation



2.1 DM/PL Language Basics

2.1.1 Introduction

In order to be fully compatible with the many features incorporated in the cutter itself, such as automatic FlexCut or recut from the cutter UI, follow these guidelines:

1. Each DM/PL file can be preceded by encapsulated commands, for example to setup the parameters for OPOS.

2. Each DM/PL file must include following commands in the initialization string:
   - The ;: Select command.
   - The ECx command.
   - The A or R command.

3. Afterward, you have the option to use some additional commands such as the Tool select Px command. This can be used multiple times within a file. It is also possible to request the size of

the loaded media, using the ER command.

4. Vector data using U and D commands follow.

5. An end of file command is strongly recommended to terminate the file. The end of file commands are: @ or e commands. **The e command is recommended.**
If you don't want to move to the end of the cut sign, then use @ command. However if you want to move to the end of the cut sign, then use the e command. These end of file commands are necessary if you want to use the FlexCut feature, the panelling feature and the OPOS features.
With the Fx command you have more flexibility on where to move the knife at the end of the file.

2.1.2 The ;: Select Command

The Select *;:* command (semicolon double point) selects the cutter. When the cutter is selected it can receive information until it receives a deselect command (e or *@)*.

Note: *This command is required before every any other data string.*

2.1.3 The ECx Coordinate Addressing Command

The Coordinate Addressing *ECx* command sets user units to a specified resolution, raises the knife, moves it to the home position.
n selects the addressing resolution, (resolution of the x- and y-coordinates) and is an alphanumeric expression 0, 1, 5, M, or N (see Table 2.1 - Coordinate Addressing).

Command	User Resolution
EC1	0,001 inch
EC5	0,005 inch
ECM	0,1 mm
ECN	0,025 mm

Table 2.1 - Coordinate Addressing

Note: *This command is required at the start of the data string.*

2.1.4 The A Absolute Addressing Command

The Absolute addressing *A* command selects the absolute addressing mode. In this mode, all x- and y-coordinates are with respect to the origin at the lower left corner of the chart. The device remains in this mode until it receives a Relative addressing *R* command or a deselect command (*e* or *@)*.

Example: `;: EC1 A U 5000,5000 D 2000,2000 e`

In this example, the device is selected with the ;: command and 1/1000 inch resolution is selected with the *EC1* command. Absolute knife positioning is selected with the *A* command. The knife is sent in up position to absolute coordinate 5000,5000. The knife is lowered with the down *D* command and a line is cut to absolute coordinate 2000,2000. The device is deselected with a deselect *e* command.

Note: *If no addressing mode is specified before any vector data is processed the Absolute addressing mode will be used by default.*

2.1.5 The R Relative Addressing Command

The Relative Addressing *R* command selects the relative addressing mode. In this mode, all x- and y-coordinates are relative to the present knife position. The device remains in this mode until it receives an Absolute Addressing *A* command or a deselect command (*e* or *@)*.

Example: `;: EC1 R U 5000,5000 D 2000,2000 e`

In this example, the device is selected with the *;:* command and 1/1000 inch resolution is selected with the Coordinate Addressing *EC1* command. Relative knife positioning is selected with the *R* command. The knife is sent to relative coordinate 5000,5000. The knife is lowered and a line is cut to relative coordinate 2000,2000 (this is absolute coordinate 7000,7000). The device is deselected with the deselect *e* command.

Note: *If no addressing mode is specified before any vector data is processed the Absolute addressing mode will be used by default.*

2.1.6 The D Down Command

The Down *D* command lowers the tool. The tool remains in the down position until an Up *U*, Coordinate Addressing *EC*, Frame *F* or End of Plot *e* command is received.

2.1.7 The U Up Command

The Up *U* command raises the tool.

2.1.8 The x,y Vector Move Command

This command causes the tool to move to a new user position specified by coordinate *x,y*. This new position depends on the present Absolute Addressing *A* or Relative Addressing *R* tool positioning command in effect, and the current user unit (see 2.1.3 The ECx Coordinate Addressing Command). In Absolute Addressing *A* mode, the new position is x,y user units from the present origin. In Relative Addressing *R* mode, the new position is *x,y* user units from the present position.

Example: `;: ECM A D 0,1000 1000,1000 1000,0 0,0 U e`

In the above sample, the cutter is selected, set to 0.1mm resolution, absolute addressing with the *ECM* command, then the tool is lowered and a square of 10 centimetres is cut. The tool is then raised and the device is deselected with the *e* command.

Note: *All coordinates MUST be integer values only.*

2.1.9 The e End Of Plot Command

For roll media the origin is moved several mm's beyond the last sign, more precise, beyond the last down vector. The amount it moves behind the last cut sign is defined with the RECUT_OFFSET parameter (can be set via encapsulated language or via the cutter UI). The end of plot command also deselects the cutter. If new data needs to be sent, it must be preceded by the ;: select command.

The recut command (encapsulated or through keypad) will work correctly when using this command.

2.1.10 The ce Cut-off with End Of Plot Command

The S class 3 cutter has a “cut-off knife” to cut off a sheet out of a roll. Sending the *ce* command will cut off the media while doing the end of plot command. To optimize the media usage, the RECUT_OFFSET will be used in combination with the CUTMEDIA_OFFSET.

Note: *The recut offset should be larger than the cut-off margin if both parameters are used together. For OPOS jobs, using the cut-off command, it is required that the recut offset is at least 30mm bigger than the cut-off margin.*

2.1.11 The @ Deselect Command

The Deselect @ command sets the cutter deselected. After the device receives the command, it continues to process the data that is already in the buffer and does not accept new data until it is selected again.
The deselect command does not affect any of the device’s parameters, such as the resolution, origin, etc…

The recut command (can be set via encapsulated language or via the cutter UI) will work correctly when using this command.

2.1.12The BPn Tool Pressure Command

The Tool Pressure *BPn* command sets a desired pressure. The argument n specifies the pressure in grams. The numeric range for n is an integer from 0 to the maximum pressure, which depends on the device (see 8.1 Maximum Pressure and Speed by Model).

2.1.13 The BOn Overcut Command

The Overcut *BOn* command sets the desired overcut. The argument n specifies the length of the overcut in 0,1mm. The numeric range for n is an integer from 0 to 10.

2.1.14 The BMn Multipass Command

The Multipass *BMn* Command sets the amount of times the same cutline will be cut. The numeric range for n is an integer from 1 to 7.
On receiving a tool select *Px* command, this parameter is reset to the value stored in the MULTIPASS parameter (can be set via encapsulated language or via the cutter UI).

2.1.15The Fn Frame Command

The Frame *Fn* command is used to move the origin in X-axis direction. Upon receipt of this command the tool is raised and the media is advanced to the new home position.
n specifies the location of the new home position. This parameter is expressed in user units as set by the present coordinate addressing *EC* command. *n* can be positive or negative. If *n* is zero, the present position is defined as the new home position, but the media does not advance.

Example: `;: EC1 F1000`

With *EC1* (see 2.1.3 The ECx Coordinate Addressing Command) the coordinate addressing is set to 1/1000 inch, the *Fn* command sets the new home position 1000 user units (= 1 inch) from the present one, and advances accordingly the media with 1 inch.

2.1.16The Px Tool Select Command

The Tool Select *Px* Command selects a given tool. The cutter will pause at execution of the command, and allows manual selection of the tool by the operator if the selected tool is different from the one being used.

The tool commands P6 or P10 (P6 is recommended) turn on the FlexCut option, no intervention of the operator will be needed.

For the PRO CAM head, that incorporates a tool and an extra tool, no operator intervention will be needed when switching between knife (P1) and the extra pen (P9) for example.

Following table describes the tool number necessary to select a certain tool, depending on the cutter-head that the cutter is equipped with.

	Tool selected on machine equipped with:		
Tool selection command	**Drag head**	**Tangential head**	**PRO CAM head**
P100 U		Use current installed tool	
P0	Drag knife	Tangential knife	Tangential knife
P1	Drag knife	Tangential knife	Tangential knife
P2	Pen	Pen	Pen
P6	FlexCut(1) mode with knife	FlexCut(1) mode with knife	FlexCut(1) mode with knife
P7	x	x	Extra pen (P9 preferred)
P8	Drag knife	Drag knife	Drag knife
P9	x	x	Extra pen
P10	FlexCut(1) Basic mode with knife	FlexCut(1) Basic mode with knife	FlexCut(1) Basic mode with knife
P11	x	Tangential knife	Tangential knife
P12	x	x	Extra creaser

Table 2.2 - DM/PL Tool Select Command

(1) Cutting through media, see Chapter 6

Note: *Speed, pressure, overcut and multipass are reset on receiving a Tool Select Command.*

2.1.17 The Vn Velocity Command

The Velocity *Vn* command sets the axial velocity of the tool (down position speed) for cutting. The integer argument n specifies the tool down velocity of the device and depends on the current coordinate addressing:

- If the present Coordinate Addressing is *EC0*, *EC1* or *EC5*, then the units are in inches per second.

  Example: `;: EC1 A V5 U 0,0 D 0,2500 2500,2500 2500,0 0,0 U e`

In the above sample, the cutter is selected, set to 1/1000-inch resolution, absolute addressing, a velocity of 5 inches per second with the *V5* command, then the tool is lowered and a square of 2.5 inches is cut. The tool is then raised and the device is deselected with the *e* command.

- If the present Coordinate Addressing is *ECM* or *ECN*, the units are in centimeters per second.

  Example: `;: ECM A V50 U 0,0 D 0,2500 2500,2500 2500,0 0,0 U e`

  In this sample, the cutter is again selected, the resolution is now set to 0.1 millimeter, and the velocity is then set to 50 centimeter per second with the *V50* command.

2.1.18 The ER Report Command

The Report *ER* command allows the device to send its present status to the computer. **It is mainly used to retrieve the size of the inserted media.**

When the device processes a Report *ER* command, the following information is transmitted:

- Status Byte One indicates the number of the last selected pen/knife, the present up or down status of the pen/knife, whether or not the present location of the knife on the cutting surface is inside the window limits, and the present full or half chart format (large or small). See also Table 2.4 - ER Report Format. Note that bit seven is always set to 0.

7	*6*	*5*	*4*	*3*	*2*	*1*	*0*
0	0 = Large Chart 1 = Small Chart	0 = Inside Window 1 = Outside Window	0 = Pen Up 1 = Pen Down	Pen Number (MSB)	Pen Number	Pen Number	Pen Number (LSB)

Table 2.3 - Status Byte One

- Status Byte Two is not used (Reserved).
- The remaining bytes indicate: the present x-, y-coordinate position of the knife, the present window limit coordinates, and the present viewport limit coordinates.

Note: *All coordinates are in user units (see 2.1.3 The ECx Coordinate Addressing Command) and relative to the origin.*

The report information is sent in ASCII BCD (binary-coded decimal) format. The values that make up the status string are decimal integers. The digits of each decimal integer are represented by their ASCII equivalents. It is the responsibility of the receiving software to convert the ASCII representations of the decimal integers to binary decimal integers before performing any mathematical operations on them. The report is always enclosed within parentheses () and followed by a carriage return. Commas separate the values. The sign of each x-, y-coordinate immediately precedes the coordinate value. A space indicates a positive value, while a minus (-) indicates a negative value. Each status byte is three digits in length. The x,y coordinate length is 7 digits. Leading zeros are used as required to pad the x-, y-coordinate lengths to their appropriate lengths. The total string length is 100 characters.

Summary of data sent :

(	StatusByte 1	Status byte2	Current X pos	Current Y pos	Window Lower Left X	Window Lower Left Y	Window Upper Right X	Window Upper Right Y	Viewport Lower Left X	Viewport Lower Left y	Viewport Upper Left X	Viewport lower Left X	)	CR

Table 2.4 - ER Report Format

Example : `;:` `ECN ER`

In this example, the device is selected with the In this example, the device is selected with the *;:* command and 0.025mm resolution is selected with the Coordinate Addressing *ECN* command. The cutter is then commanded to send a report with the *ER* command.

ER returns:
`(017,084, 0001000, 0002000, 0000000, 0000000, 2000000, 0014650, 0000000, 0000000, 2000000, 0014650)<CR>`

In this report we can analyse the following:

- Status Byte One (017) shows that tool 1 was the last requested tool and it is presently in the down position inside the present window limits.
- Status Byte Two (084) is reserved and not used.
- Current X Position (0001000) : the present tool-position x-coordinate is 1000 * 0.025mm = 25 mm.
- Current Y Position (0002000) : the present tool-position y-coordinate is 2000 * 0.025 mm = 50 mm.
- Window Lower Left X (0000000) defines the lower left x-coordinate of the window at 0.
- Window Lower Left Y (0000000) defines the lower left y-coordinate of the window at 0.
- Window Upper Right X (2000000) defines the upper right x-coordinate of the window at 2000000 * 0.025mm = 50000mm = 50m.
- Window Upper Right Y (0014650) defines the upper right x-coordinate of the window at 14650 * 0.025mm = 366.25mm.
- Viewport Lower Left X (0000000) defines the lower left x-coordinate of the viewport at 0.
- Viewport Lower Left Y (0000000) defines the lower left y-coordinate of the viewport at 0.
- Viewport Upper Right X (2000000) defines the upper right x-coordinate of the viewport at 2000000 * 0.025mm = 50000mm = 50m.
- Viewport Upper Right Y (0014650) defines the upper right x-coordinate of the viewport at 14650 * 0.025mm = 366.25mm.
- A carriage return <*CR*> terminates the report.

Summarized we can say that a knife was selected and that it is in the down position, at x-y location 25mm and 50mm from the origin. There is media loaded which is 366.25mm wide and 50m long. As this is the available media-size, actual media-size may differ, because we need some space for the pinch-rollers to hold the media. There is also an assumption that a full roll of vinyl was present.

2.1.19The EWx Job Length Command

The job length *EWx* sets the length of the job, x specifies the length of the job. This parameter is expressed in user units as set by the coordinate addressing EC command. The EW command should be sent before any cut data and after the addressing command.
When the cutter receives this command, it will pre-load the distance x. The Report ER command can be used to check the distance that could be loaded.

This command should not be used in combination with panelling as it would cancel out the benefits of panelling (The biggest advantage of using panelling is that it does not preload the complete job).

Example: ;: EC1 EW40000

With *EC1* (see 2.1.3 The ECx Coordinate Addressing Command) the coordinate addressing is set to 1/1000 inch, the EW command will unroll the media so that a job of 40000 user units (= 40 inch) fits on the unrolled media.

2.1.20Sample DM-PL File

```
;: EC1 A U 1000,1000 D 2000,2000 2000,0 e
```

In this example, the device is selected with the *;:* command and 1/1000 inch resolution is selected with the Coordinate Addressing *EC1* command. Absolute knife positioning is selected with the *A* command. The knife is sent in up position to absolute coordinate *1000,1000*. The knife is lowered with the Down *D* command and a line is cut to absolute coordinate *2000,2000.* Then a line is cut to position *2000,0*. The media is moved to the end of the cut area by the *e* command, so that the next sign will not be placed above the current one. This command also deselects the device.

2.2 HP-GL Language Basics

Only a subset of HP-GL and HP-GL/2 is presented in this document and supported by our cutters.
There is no need for an select command, most often HP-GL files start with an *IN;* or *BP;* command.
The standard resolution for HP–GL is 0.025 mm and all commands must end with a semicolon *;*. It is strongly recommended to end a file with a PG; command, otherwise functions like recut, FlexCut, panelling or OPOS barcode will not work.

2.2.1 The IN; Initialize Command

The *IN;* initialize command resets all parameters that were changed by HP-GL commands.

2.2.2 The BP; Begin Plot Command

The *BP;* Begin Plot command resets all parameters that were changed by HP-GL/2 commands and moves the origin to the end of the previous cut sign.

2.2.3 The PAx,y; Absolute Addressing Command

The Absolute Addressing *PA;* command selects the absolute addressing mode. In this mode, all x-, y-coordinates are with respect to the origin at the lower left corner of the chart. The device remains in this mode until it receives a Relative Addressing *PR* command.

Example: `IN; PA; PU 5000,5000;PD 2000,2000;`

In this example, The cutter is initialized with the *IN;* command. Absolute knife positioning is selected with the *PA;* command. The knife is sent in up position to absolute coordinate 5000,5000. The tool is lowered with the Down *PD* command and a line is cut to absolute coordinate 2000,2000.

2.2.4 The PRx,y; Relative Addressing Command

The Relative Addressing *PR;* command selects the relative addressing mode. In this mode, all x-, y-coordinates are relative to the present tool position. The device remains in this mode until it receives an Absolute Addressing *PA* command.

2.2.5 The PDx,y; Down Command

The Down *PD;* command lowers the knife. This command causes the knife to move to a new user position specified by coordinate x,y. This new position is dependent on the present Absolute *PA* or Relative *PR* Addressing command in effect. In Absolute Addressing *PA* mode, the new position is x,y user units from the present origin. In Relative Addressing *PR* mode, the new position is x,y user units from the present position.

2.2.6 The PUx,y; Up Command

The Up *PU;* command raises the knife. This command causes the knife to move to a new user position specified by coordinate x,y. This new position is dependent on the present Absolute *PA* or Relative *PR* Addressing command in effect. In Absolute Addressing *PA* mode, the new position is x,y user units from the present origin. In Relative Addressing *PR* mode, the new position is x,y user units from the present position.

2.2.7 The VSn; Velocity Command

The Velocity *VS;* Command sets the down velocity. n specifies the velocity of the device in cm/s.

2.2.8 The SPn; Tool Select Command

The Tool Select *SPn* Command selects a given tool. The cutter will pause at execution of the command, and allows manual selection of the tool by the operator if the selected tool is different from the one being used.

The tool commands P6 or P10 (P6 is recommended) turn on the FlexCut option, no intervention of the operator will be needed.

For the PRO CAM head, that incorporates a tool and an extra tool, no operator intervention will be needed when switching between knife (P1) and the extra pen (P9) for example.

Following table describes the tool number necessary to select a certain tool, depending on the cutter-head that the cutter is equipped with.

	Tool selected on machine equipped with:		
Tool selection command	**Drag head**	**Tangential head**	**PRO CAM head**
P100 U		Use current installed tool	
P0	Drag knife	Tangential knife	Tangential knife
P1	Drag knife	Tangential knife	Tangential knife
P2	Pen	Pen	Pen
P6	FlexCut(1) mode	FlexCut(1) mode	FlexCut(1) mode with tangential knife
P7	x	x	Extra pen (P9 preferred)
P8	x	Drag knife	Drag knife
P9	x	x	Extra pen
P10	FlexCut(1) Basic mode	FlexCut(1) Basic mode	FlexCut(1) Basic mode with tangential knife
P11	x	Tangential knife	Tangential knife
P12	x	x	Extra creaser

Table 2.5 – HP-GL Tool Select Command

(1) Cutting through media, see Chapter 6

Note: *Speed, pressure, overcut and multipass are reset on receiving a Tool Select Command.*

2.2.9 The FSn; Force Select Command

The Force Select FSn; command sets the tool pressure. It is expressed in grams. The numeric range for n is an integer from 0 to the maximum pressure that depends on the device (see 8.1 Maximum Pressure and Speed by Model).

2.2.10 The OVn; Overcut Command

The Overcut *OVn* command sets the desired overcut. The argument n specifies the length of the overcut in 0,1mm. The numeric range for n is an integer from 0 to 10.

2.2.11 The MPn; Multipass Command

The Multipass *MPn* Command sets the amount of times the same cutline will be cut. The numeric range for n is an integer from 1 to 7.
On receiving a tool selected command, this parameter is reset to the value stored in the MULTIPASS parameter (can be set via encapsulated language or via the cutter UI).

2.2.12 The OH; Output Hardclip Command

The *OH;* output hardclip command returns the size of the loaded media. Data returned by the cutter:

Window Lower Left X	Window Lower Left Y	Window Upper Right X	Window Upper Right Y	Carriage Return

Table 2.6 - OH Report Format

2.2.13 The PG; End Of Plot Command

For roll media the origin is moved several mm's beyond the last down vector, so that a new sign is not cut on the previous one. The amount it moves behind the last cut sign is defined with the RECUT_OFFSET parameter.

2.2.14 The EC; Enable Cut-off Command

The S class 3 cutter has a "cut-off knife" to cut off a sheet out of a roll. Sending the *EC;* command will enable cut off while doing the end of plot command. To optimize the media usage, the RECUT_OFFSET will be used in combination with the CUTMEDIA_OFFSET.
So to cut off the media, send the design, send the *EC;* command and finally send the *PG;* command.

> ***Note:*** *The recut offset should be larger than the cut-off margin if both parameters are used together. For OPOS jobs, using the cut-off command, it is required that the recut offset is at least 30mm bigger than the cut-off margin.*

2.2.15 The EWx; Job Length Command

The *EWx;* job length command sets the length of the job, x specifies the length of the job. This parameter is expressed in user units. The EW command should be sent before any cut data.
When the cutter receives this command, it will pre-load the distance x. The *OH;* output hardclip command can be used to check the distance that could be loaded.
This command should not be used in combination with panelling as it would cancel out the benefits of panelling (The biggest advantage of using panelling is that it does not preload the complete job).

2.2.16 Sample HP-GL File

Example: `IN; PA;PU1000,1000;PD2000,2000;PD 2000,0;PG;`

In this example, the IN; resets the device. Absolute tool positioning is selected with the *PA;* command. The tool is sent in up position to absolute coordinate 1000,1000. The tool is lowered with the Down *PD* command and a line is cut to absolute coordinate 2000,2000 then a line is cut to position 2000,0. The media is moved to the end of the cut area by the *PG;* command, so that the next sign will not be placed above the current one.

3 Encapsulated Language

The cutter control commands listed below constitute the machine's native control language.
The encapsulated commands are given in the form:

COMMAND option1 | option2 | option3

Where "command" is the command to type in (including spaces and the "=" symbol) and "option1", "option2", etc. stand for the values or arguments that work with the command. You can only use one argument at a time. Alternative arguments are shown separated by "|" which is not part of the command or its argument. End each command with a period, or with a carriage return/line feed pair (generated by pressing the <Enter> key on PC-compatible computers).
Possible commands are:

```
SET item_name = option.
END.
MENU [ menu item ].
QUERY.
SET_ORIGIN.
LOAD_MARKERS.
```

The command interpreter is activated when the cutter receives the following sequence of characters:

```
<esc>;@:
```

The sequence begins with the escape character (ASCII 27 in decimal, followed by ASCII 58, 64 and 59). When activated, the command interpreter responds with "READY". It prompts you to enter commands with the ">" symbol.

```
END.
```

After accessing the machines command interpreter, send an END command to the machine to exit the command interpreter. During the dialogue with the machine, send an END command to end the dialogue session. When creating a data encapsulation file, put "END" immediately before the beginning of the data to plot. A period, rather than a carriage return/line feed pair, is the preferred way to terminate the END command in a data encapsulation file, since it provides a visible marker at the end of the encapsulation header.

Note : *After accessing the machines command interpreter, be sure to send an "END." command before sending any data to cut. Otherwise the machine will try to interpret the data as encapsulated control commands, with unpredictable results.*

3.1 Changing Parameters Settings Commands

The **SET** command is used to set the menu items. For more information about the menu items, check the cutter user manual.

3.1.1 MARKER_X_DIS = [a number in the range 1200 to 52000]

Allows for the marker y distance to be specified in 0.025 mm units.

```
<esc>;@:
SET MARKER_X_DIS=2400.
END.
```

This sample defines the X-markers 60 mm apart.

3.1.2 MARKER_Y_DIS = [a number in the range 1200 to 64000]

Allows for the marker y distance to be specified in 0.025 mm units.

```
<esc>;@:
SET MARKER_Y_DIS=21600.
END.
```

This sample defines the Y-markers 540 mm apart.

3.1.3 MARKER_X_SIZE = [a number in the range 80 to 400]

Allows for the marker x size to be specified in 0.025 mm units.
The default value is 120.

```
<esc>;@:
SET MARKER_X_SIZE=120.
END.
```

This sample defines an X-marker size of 3mm.

3.1.4 MARKER_Y_SIZE = [a number in the range 80 to 400]

Allows for the marker y size to be specified in 0.025 mm units.
The default value is 120.

```
<esc>;@:
SET MARKER_Y_SIZE=120.
END.
```

This sample defines an Y-markers size of 3mm.

Note: *The S Class 3 only supports square marks, so x size and y size should be the same.*

3.1.5 MARKER_X_N = [a number in the range 2 to 128]

Allows for the number of markers along the x axis to be specified. When the special load procedure is

called with MARKER_X_N, then the specified number of markers will be searched for.
The minimum value to be used for MARKER_X_N is 2 as a minimum of 2 markers is needed along the x axis.

```
<esc>;@:
SET MARKER_X_N=6.
END.
```

3.1.6 SPECIAL_LOAD = [OPOS,OPOS_XY,OPOS_XY2,OPOS_XTRA]

Use this to set select which kind of alignment method will be used in the print and cut workflow after the “LOAD_MARKERS” command.

Note: *The alignment method must be set by the RIP, the S Class 3 cutter does not have the option to set the alignment method through the UI.*

3.1.7 SHEET_MODE = [OFF,ON]

Enables the sheet mode. When turned on the alignment and/or cutting will be started automatically from the 2nd sheet on. The default value is OFF.

```
<esc>;@:
SET SHEET_MODE ON.
END.
```

3.1.8 PANELLING = [OFF,ON]

Defines if the machine should panel the data.

3.1.9 PANELLING_SIZE = [a number in the range 1 to 250]

Sets the panel size in cm.

3.1.10 RECUT_OFFSET = [a number in the range 0 to 4000]

Sets the offset between two job when recutting the same job in mm.

3.1.11 CUTMEDIA_OFFSET = [a number in the range 0 to 250]

Sets the offset used to cut off the media after a job in mm.

3.1.12 VELOCITY = [50,100,200,300,400,500,600,700,800,900,1000]

Use this command to specify the velocity in mm/s. The maximum speed depends on the model (see 8.1 Maximum Pressure and Speed by Model). If an invalid value is used the command will be ignored and machine velocity will not change. Accepted values for specific model can be requested by using the MENU command (see 3.3.2 MENU [item name].).

```
<esc>;@:
SET VELOCITY=600.
END.
```

3.1.13OVERCUT = [a number in the range 0 to 10]

Use this command to set overcut. One unit is about 0.1 mm.
The default value is 1.

3.1.14 OPTICUT = [ON,OFF]

Use this command to set the OptiCut.

3.1.15 TOOL = […]

For cutters with drag head
TOOL = [PEN,DRAG_KNIFE]

For cutters with tangetial head
TOOL = [PEN,DRAG_KNIFE,TANGENTIAL_KNIFE]

For cutters with tangential PRO CAM head:
TOOL = [PEN,DRAG_KNIFE,TANGENTIAL_KNIFE,EXTRA_PEN,CREASER]

Use this command to select the cutter's tool.

3.1.16FLEX_CUT = [OFF,MODE1,MODE2]

Use this to enable FlexCut mode, set to mode 1 or mode 2.

> ***Note:*** *It's recommend to use the Tool Select Command in DM/PL or HP-GL to use FlexCut. The FLEX_CUT is considered as deprecated.*

3.1.17 FULL_PRESSURE = [a number in the range 20 to 1000]

Sets the full pressure value for FlexCut mode in gram (see 8.1 Maximum Pressure and Speed by Model).

3.1.18CUT_LENGTH = [a number in the range 10 to 10000]

Sets the length value to cut at full pressure for FlexCut mode specified in 0.025 mm units.

3.1.19FLEX_PRESSURE = [a number in the range 20 to 1000]

Sets the flex pressure (reduced pressure) value for FlexCut mode in gram (see 8.1 Maximum Pressure and Speed by Model).

3.1.20FLEX_LENGTH = [a number in the range 10 to 10000]

Sets the length value to cut at flex pressure for FlexCut mode specified in 0.025 mm units.

3.1.21FLEX_VELOCITY = [50,100,200,300,400,500,600,700,800,900,1000,AUTO]

Use this command to specify the velocity when cutting through in flex-mode, this in mm/s. The maximum speed depends on the model (see 8.1 Maximum Pressure and Speed by Modell). When set to Auto, it will follow the VELOCITY settings.

3.1.22 FLEX_PANEL_SIZE = [a number in the range 1 to 250]

Sets the panel size in cm for FlexCut mode.

3.1.23 SORTING_ENABLE = [OFF,ON,START_POINT]

Defines if vector optimization is done or not. ON means that vectors are optimized for the cutting direction (media movement), while START_POINT optimizes the starting point for closed curves.

3.1.24 MULTIPASS = [a number in the range 1 to 7]

Defines how many times the same cutline will be cut. If set to n, the same line will be cut n times. Default is 1.

3.2 Executive Commands

3.2.1 SET_ORIGIN = X,Y.

This command moves the origin relatively from the current position. If no X and Y are specified the current position will be the new origin. X/Y can be either positive or negative. The unit of the coordinates is 0.025 mm.

```
<esc>;@:
SET_ORIGIN=0,6000.
END.
```

3.2.2 LOAD_MARKERS.

Allows the user to execute the “SPECIAL_LOAD” alignment method. If the LOAD_MARKERS procedure is aborted or the cutter failed to sense all the markers properly this command will return “ERROR : [error message].”
If the LOAD_MARKERS procedure properly senses all markers, “MARKERS LOADED.” will be returned.

> ***Note:*** *The LOAD_MARKERS command should always be preceded by the alignment method. Whether the command is used in the cut data or as a start command for the Barcode workflow.*

Use of LOAD_MARKERS as start of barcode workflow.

```
<ESC>;@:
SET SPECIAL_LOAD = OPOS_BARCODE.
LOAD_MARKERS.
END.
```

Use of LOAD_MARKERS in cut data.

```
<ESC>;@:
SET SPECIAL_LOAD = OPOS_XY.
SET MARKER_X_SIZE = 80.
SET MARKER_Y_SIZE = 80.
SET MARKER_X_DIS = 4400.
SET MARKER_Y_DIS = 3920.
SET MARKER_X_N = 2.
```

```
LOAD_MARKERS.
END.
```

3.2.3 RECUT = n.

Recuts the last job n times.

```
<esc>;@:
RECUT 12.
END.
```

3.3 Response Commands

Response commands are used to elicit information from the cutter. Response commands are useful only during dialogues with the cutter's command interpreter, in which the cutter's responses can be received. See Dialogue Example below for an example of their use.

3.3.1 MENU.

When the command interpreter receives the MENU command by itself, the interpreter responds with the following information:

item count ITEMS-
item name1 : type = current value
item name2 : type = current value
...
where "item count" is the number of configuration settings: "item name1", "item name2", etc. are the names of the settings; "type" is the type of value allowed for each setting; and "current value" is the current value for each setting.

Send the following to the cutter:

```
<esc>;@:
MENU.
END.
```

This could result in the following prompt, dependent of the capabilities of the connected cutter:

```
READY.

>54 ITEMS-
  KNIFE_PRESSURE : numeric{0..600} = 50
    PEN_PRESSURE : numeric{0..600} = 80
     DRAG_OFFSET : numeric{0..100} = 43
        VELOCITY : enumtext{50,100,200,300,400,500,600,700,800,900,1000} = 800
         OVERCUT : numeric{0..10} = 0
  CONCATENATION : numeric{0..20} = 0
         OPTICUT : enumtext{OFF,ON} = OFF
       SMOOTHING : enumtext{OFF,ON} = OFF
       EMULATION : enumtext{DMPL,HPGL,HPGL_2,AUTO} = AUTO
            TOOL : enumtext{BALLPOINT,T_DRAG_KNIFE,TANGENTIAL_KNIFE,POUNCER} = T_DRAG_KNIFE
     MENU_UNITS : enumtext{ENGLISH,METRIC} = METRIC
 DMPLADDRESSING : enumtext{EC1,EC5,ECN,ECM,EC0} = ECN
    HPGL_ORIGIN : enumtext{CENTER,RIGHT_FRONT} = RIGHT_FRONT
      BAUD_RATE : enumtext{2400,4800,9600,19200,38400,57600} = 38400
         PARITY : enumtext{NONE,MARK,EVEN,ODD} = NONE
        RTS/DTR : enumtext{TOGGLE,HIGH} = TOGGLE
  TOOL_COMMANDS : enumtext{ACCEPT,IGNORE} = ACCEPT
       AUTOLOAD : enumtext{ON,OFF,ASK} = ON
    SPECIAL_LOAD : enumtext{OPOS,XY_ADJUST,XY_ALIGN,X_ALIGN,OPOS_XY,OPOS_BARCODE,FORCE_OPOSXY} = OPOS
OPOS_SHEET_MODE : enumtext{OFF,ON} = OFF
       FLEX_CUT : enumtext{OFF,MODE1,MODE2} = OFF
  FULL_PRESSURE : numeric{0..600} = 70
     CUT_LENGTH : numeric{10..10000} = 144
```

```
   FLEX_PRESSURE : numeric{0..600} = 30
     FLEX_LENGTH : numeric{10..10000} = 19
   FLEX_VELOCITY : enumtext{50,100,200,300,400,500,600,700,800,900,1000,AUTO} = AUTO
      CONFIGUSER : numeric{1..8} = 1
     MEDIA_SENSE : enumtext{OFF,ON,FRONT_OFF} = FRONT_OFF
    MARKER_X_DIS : numeric{1200..52000} = 2400
    MARKER_Y_DIS : numeric{1200..64000} = 21600
   MARKER_X_SIZE : numeric{48..400} = 80
   MARKER_Y_SIZE : numeric{48..400} = 80
      MARKER_X_N : numeric{2..128} = 14
    40G_PRESSURE : numeric{0..250} = 26
   400G_PRESSURE : numeric{0..250} = 200
    DRAG_LANDING : numeric{0..250} = 17
     PEN_LANDING : numeric{0..250} = 17
    TANG_LANDING : numeric{0..250} = 17
    RECUT_OFFSET : numeric{0..4000} = 0
 CUTMEDIA_OFFSET : numeric{0..255} = 0
        LANGUAGE : enumtext{ENGLISH,FRENCH,GERMAN,SPANISH,ITALIAN,DUTCH,POLISH,CZECH} = ENGLISH
  POUNC_PRESSURE : numeric{0..600} = 120
       POUNC_GAP : numeric{1..50} = 1
    UP_VELOCITY : enumtext{50,100,200,300,400,500,600,700,800,900,1000,AUTO} = AUTO
UP_ACCELERATION_ : enumtext{1,2,3,5,10,20,25,30,35,40,AUTO} = AUTO
DOWN_ACCELERATION_ : enumtext{1,2,3,5,10,20,25,30,35,40,AUTO} = AUTO
 VEL_LONG_VECTOR : enumtext{AUTOMATIC,NORMAL} = AUTOMATIC
        TURBOCUT : enumtext{QUALITY,SPEED} = SPEED
  QUALITY_TUNING : enumtext{OFF,ON} = ON
         ROLL_UP : enumtext{OFF,END_OF_JOB,END_OF_PANEL} = OFF
       PANELLING : enumtext{OFF,ON} = OFF
  PANELLING_SIZE : numeric{0..250} = 50
    PANEL_REPLOT : numeric{0..99} = 0
  SORTING_ENABLE : enumtext{OFF,ON} = OFF

     >
```

3.3.2 MENU [item name].

When you add the name of a setting (item name) to the MENU command, the command interpreter responds with the type of value and the current value for the setting named:

item name: type = current value

Send this:

```
<esc>;@:
MENU VELOCITY.
END.
```

To receive this :

```
READY

>
   VELOCITY : enumtext{50,100,200,300,400,500,600,700,800,900,1000} = 600
>
```

3.3.3 QUERY.

When you send the QUERY command, the command interpreter responds with the cutter's model name and ROM number.

Send the following to the cutter:

```
<esc>;@:
QUERY.
END.
```

You will receive something like this :

```
>
S3T160
9987005 9987005
>
```

3.4 Encapsulated File Example

The following is an example of a data encapsulation file header, placed before a plot. The encapsulation header contains cutter control commands that configure the cutter for a plot following the header. Comments in *italics* are not part of the example file.

```
<esc>;@:                        Initialize encapsulation
SET VELOCITY = 600.             Set velocity to 600 mm/s
END.                            Terminate encapsulation
```

> ***Note:*** *In this example, a period, rather than a carriage return/line feed pair, has been used to terminate a command. The init-string doesn't need a period.*

4 OPOS Outline Cutting

4.1 Introduction

Summa supports several systems for aligning media in their cutters. These alignment systems are used to do the contour cutting of printed designs.
This chapter first describes shortly the principle of the OPOS alignment systems and which support the sign-making software should include.
The second part describes in detail how to implement the OPOS alignment system in the sign-making software.

4.2 Contour Cutting Problems

Depending on the selected alignment method, the cutters can counterbalance the following irregularities:

1. Rotated Design

If the printed design is not loaded straight into the unit, the contour can be rotated equally to fit the printed graphic.

2. Skewed Design

If the X and Y-axes of the printed design are not perpendicular, the contour can be skewed to fit the printed design.

3. Incorrectly Scaled Design

If the print size is different from the original design in your software due to media expansion or shrinkage, or due to printing inaccuracies, the contour can be scaled to fit the printed graphic.

Note: *The scaling can only be adjusted by a few percent.*

Any combination of the three above irregularities can be handled too. The encapsulated parameter SPECIAL_LOAD determines which alignment method is used.

4.3 The OPOS Alignment Method

The S Class 3 cutter has an option called OPOS. This accurate Optical POsitioning System guarantees precise contour cutting. The registration markers are read automatically with an optical system, which can be a sensor or a camera.

The basic idea for printing and cutting with OPOS is described hereafter in several steps:

1. Create a design for printing and cutting.
2. Add registration markers to the design. The sign-making software should facilitate this procedure for the user. The software should automatically place the markers at the correct place and with the correct size.
3. Print the design together with the markers.
4. Insert the design in the cutter.
5. Send commands to the cutter, which inform the cutter about the amount, the size and the position of the markers.
6. Start the OPOS load procedure on the cutter; this procedure will register the markers. The software can initiate this procedure.
7. The next step will be for the user to manually position the optical sensor near the first marker.
8. Finally the software must send the outline of the design to the cutter (in the DM/PL or HP-GL vector language). The cutter will then cut the design precisely and compensate errors from alignment, calibration differences between printer and cutter, rotation of media and skewing.

> ***Note:*** *The information that the cutter requires (info about markers (5), starting the OPOS load procedure (6) and the cutting data (8)) can all be sent at once to the cutter after the printed design is inserted in the cutter.*

In this document there are several references to the X and Y-axis. With the X-axis is meant the axis in which the media moves, the Y-axis correspond to the movement of the cutting head. This is the same coordinate system as for the DM/PL and HP-GL vector language used for the cutting data. The origin of the cutter is at the lower right corner of the media and corresponds to coordinate X=0 and Y=0.

4.4 Registration marks

To have full advantage of the OPOS alignment system the placement of the markers should be done fully automatically by the software. The software should have an option that places all the necessary marks for the current design. The software should then also automatically send the info about the marks to the cutter.

4.4.1 Shape

The marks must be black rectangles, all of the same size.

4.4.2 Size

It is advised to use square marks of 3 mm by 3 mm. This means that the X-size is 3 mm and the Y-size is 3 mm. Do not use marks smaller than 2 mm and do not use marks bigger than 5 mm.

4.4.3 Position

Make sure there is a white margin of about 3-4 times the mark size around the mark. If anything is printed within this margin the sensor may have trouble to locate the marks.
The first mark must be positioned at the origin of the drawing, at coordinate (0,0). More precisely the lower right corner of the mark must be at coordinate (0,0). All cutting data must then be related to the lower right corner of the mark.

Note: *Make sure there is at least 1 cm, preferably 2 cm margin between the marks and the edge of the media on the left, right and front side. For the rear edge of the media there should be at least 4 cm of margin.*

Exactly 2 rows of marks along the Y-axis are necessary. The first row includes the origin mark and is placed on the right side of the design along the Y-axis. The Y-coordinate of the right side of these marks must be 0 (see illustration 4.1 - OPOS Marks Layout).
The second row is also placed along the Y-axis at position Y-distance at the left side of the picture. The right side of these marks must be at position Y-distance (see illustration 4.1 - OPOS Marks Layout).
The Y-distance between the marks can go up to 1600 mm. This is the widest media that a cutter from Summa can accommodate.

There are 2 or more marks along the X-axis necessary. The distance between 2 marks along the X-axis (called X-distance) must be the same for all marks. The advised X-distance is 400 mm. Do not go beyond 1000 mm.
The distance between 2 marks is measured from lower right corner to lower right corner in both X and Y direction as shown in.

It is advised to put marks till the end of the drawing (in the X-direction). The last mark however does not have to be past the design. The OPOS alignment system will extrapolate the information as needed.

Note: *It can be useful to place a sign somewhere indicating the origin mark. This is useful when the printed design needs to be loaded in the cutter. Also make sure there is enough white space around every mark.*

Illustration 4.1 - OPOS Marks Layout

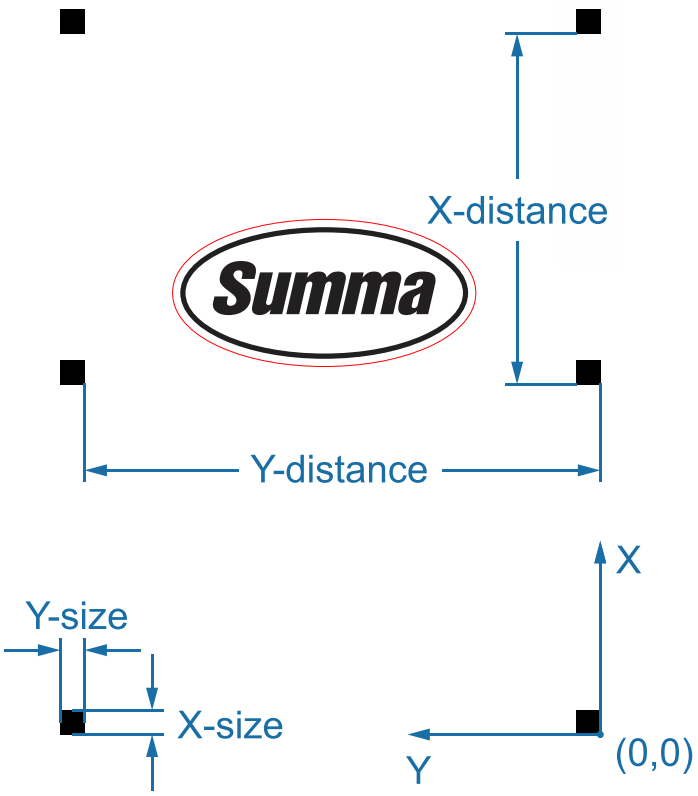
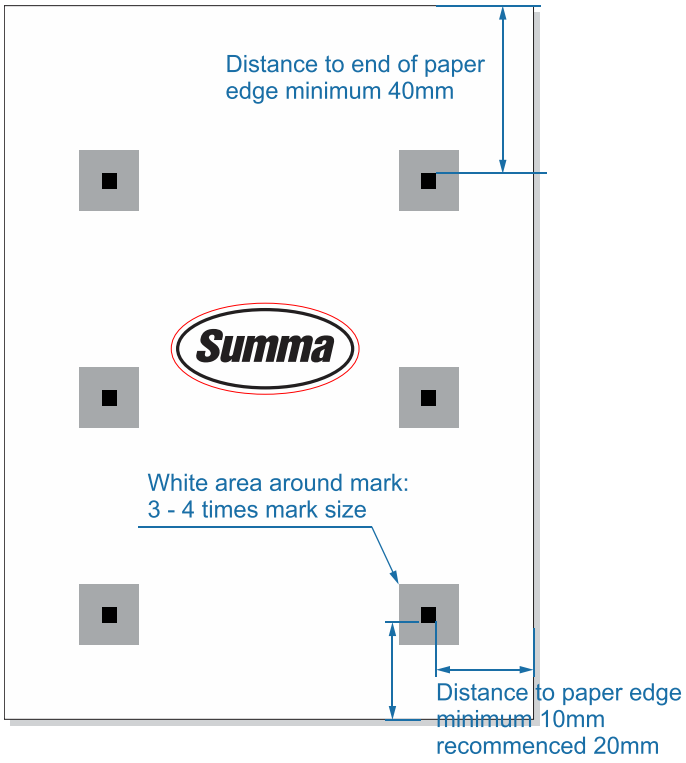


Illustration 4.2 - OPOS Marks placement



4.4.4 OPOS XY

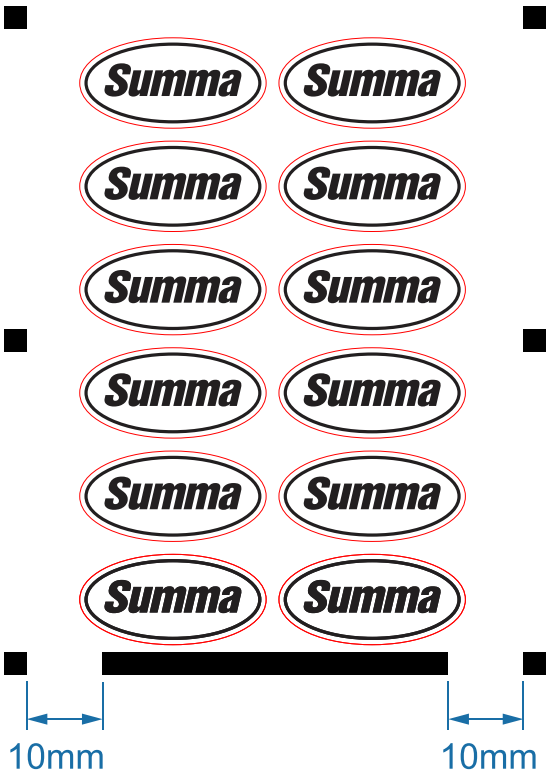
The S Class 3 cutter also supports an extra correction along the Y-axis, this feature is called OPOS XY. It will compensate for misalignment between what is printed and what is cut along the Y-axis due for example for curved print-outs along this axis.

In order to do this, a line must be added along the Y-axis between or slightly beyond the first markers.

The left and right margin between the line and the markers should be 10 mm for optimal sensing.

The thickness of the line should be 3mm.

Illustration 4.3 - OPOS XY



The OPOS XY is an extra alignment mode. So it can be set by means of the encapsulated parameter SPECIAL_LOAD.

```
[ESC];@:
SET SPECIAL_LOAD OPOS_XY.
END.
```

When enabled, the line will automatically be sensed several times in function of the width of the job.

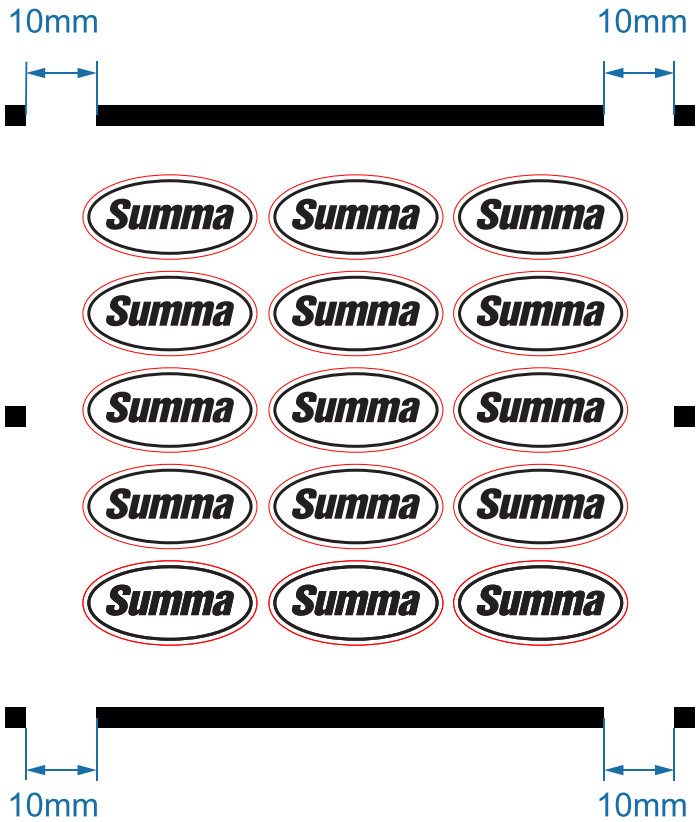
4.4.5 OPOS XY2

A first enhancement for OPOS XY is called OPOS XY2. If a line is printed between the two top marks, then this line can be scanned also, this feature is called OPOS XY2. The combination with the front and back XY-line, allows the system to contour cut longer jobs more accurately. It will compensate for misalignment between what is printed and what is cut along the Y-axis due for example for curved print-outs along this axis, and adjust along the X axis if the curvature is not the same at the bottom as at the top.

In order to do this, a line must be added along the Y-axis between the last marks.

The left and right margin between the line and the markers should be 10 mm for optimal sensing.

Illustration 4.4 - OPOS XY2



The OPOS XY2 is an extra alignment mode. So it can be set by means of the encapsulated parameter SPECIAL_LOAD.

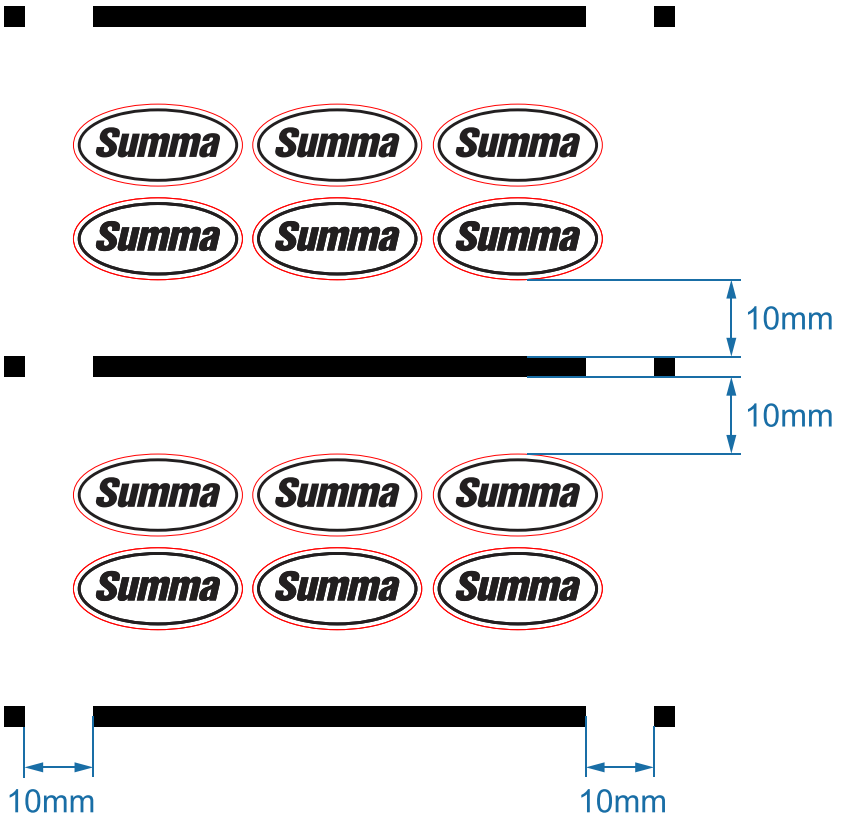
```
[ESC];@:
SET SPECIAL_LOAD OPOS_XY2.
END.
```

When enabled, the front and rear line will automatically be sensed several times in function of the width of the job. OPOS XY2 is automatically converted to OPOS XY if panelling is used.

4.4.6 OPOS XTRA

The newest option for compensating misalignment along the Y axis is OPOS XTRA. This is again an enhancement of the OPOS XY function that allows the system to contour cut longer jobs with even more accuracy.
It will compensate for misalignment between what is printed and what is cut along the Y-axis due for example for curved print-outs along this axis, and adjust along the X axis if the curvature changes in the printing direction.
For this option it is required to print a line (analogue to the OPOS XY line) between the left and right OPOS mark. All these lines will be scanned be sensed several times in function of the widths of the sign in order to compensate.
The left and right margin between the line and the markers should be 10 mm for optimal sensing.

Illustration 4.5 - OPOS XTRA



The OPOS XTRA is an extra alignment mode. So it can be set by means of the encapsulated parameter SPECIAL_LOAD.

```
[ESC];@:
SET SPECIAL_LOAD OPOS_XTRA.
END.
```

When enabled, the lines will automatically be sensed several times in function of the widths of the sign.

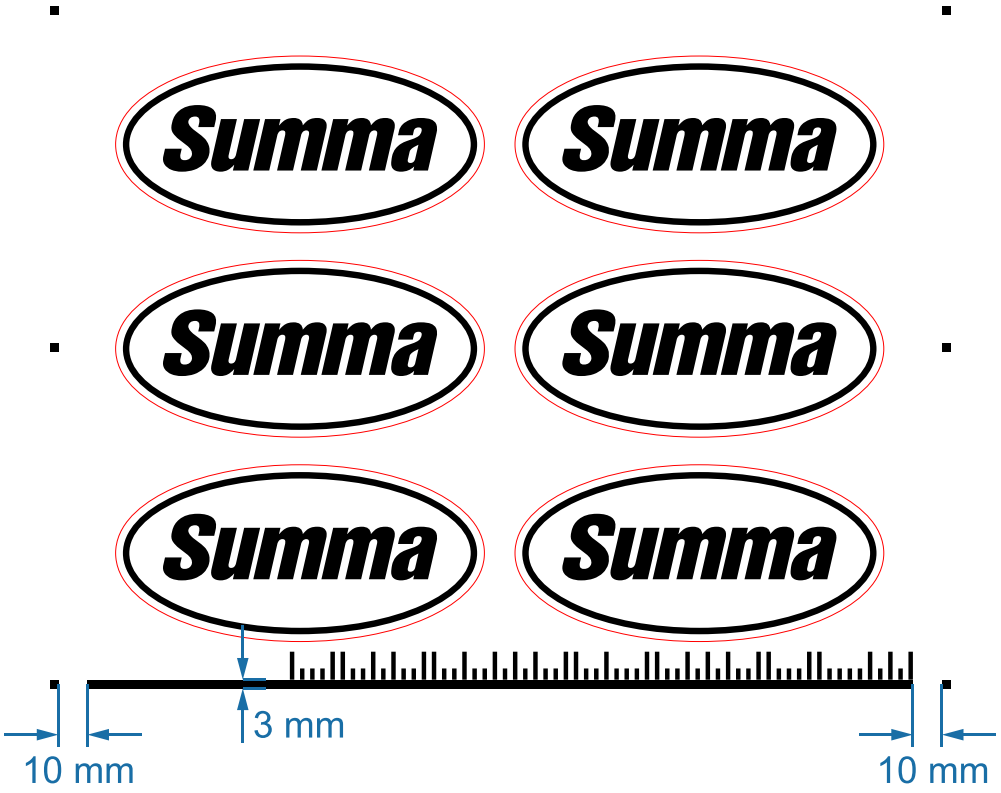
Note: *Because of its definition this is an OPOS option that will be used when using multiple copies in the RIP.*

4.4.7 OPOS BARCODE

The OPOS system is also capable of reading a barcode in order to identify the job. This way multiple jobs can be processed after each other without operator intervention. See section 4.6 on A for more info on how to use OPOS BARCODE in a workflow.

The Summa barcode consists of a line along the Y-axis with a POSTNET code on top of it. This line should be 3mm thick, aligned to the bottom of the markers and leave 10mm of white space between the markers and this line. On wider jobs this line will also be used as OPOS XY-line. On top of this line the barcode must be placed aligned to the right nearby the marker indicating the origin.

Illustration 4.6 - Basic OPOS BARCODE



The barcode is based upon 'Postnet' used by the US post (http://en.wikipedia.org/wiki/POSTNET).

Summarized definition:

Illustration 4.7 - US Post Barcode

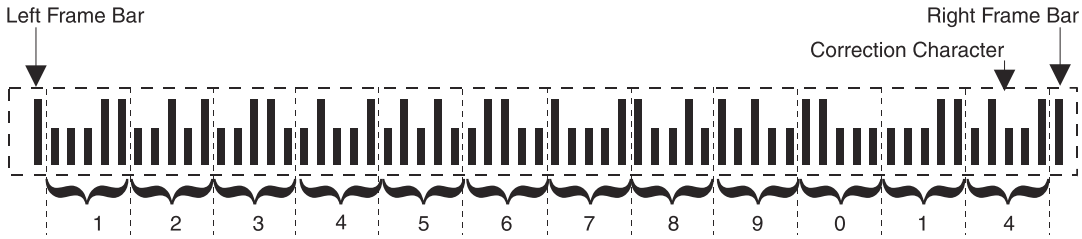


Illustration 4.8 - Barcode values

Numeric Value	Binary Code Value 7 4 2 1 0	Barcode Value 74210
1	00011	
2	00101	
3	00110	
4	01001	
5	01010	
6	01100	
7	10001	
8	10010	
9	10100	
0	11000	

- The first and last full bars in a barcode—the frame bars—do not count.
- Each digit (numeric value) is represented by five bars.
- Value is 11 digit number.
- The last five bars in the barcode make up the correction character. All barcodes, when added together, must equal a multiple of 10.

The OPOS BARCODE should be three times the official size and placed on a line.

Illustration 4-9 - Size of OPOS Barcode Digit

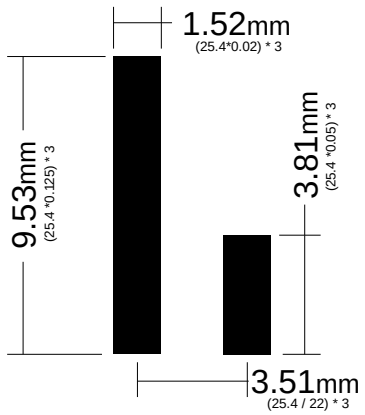
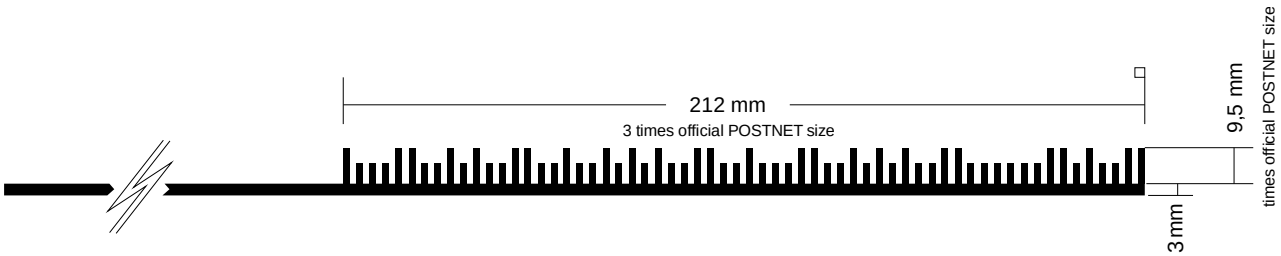


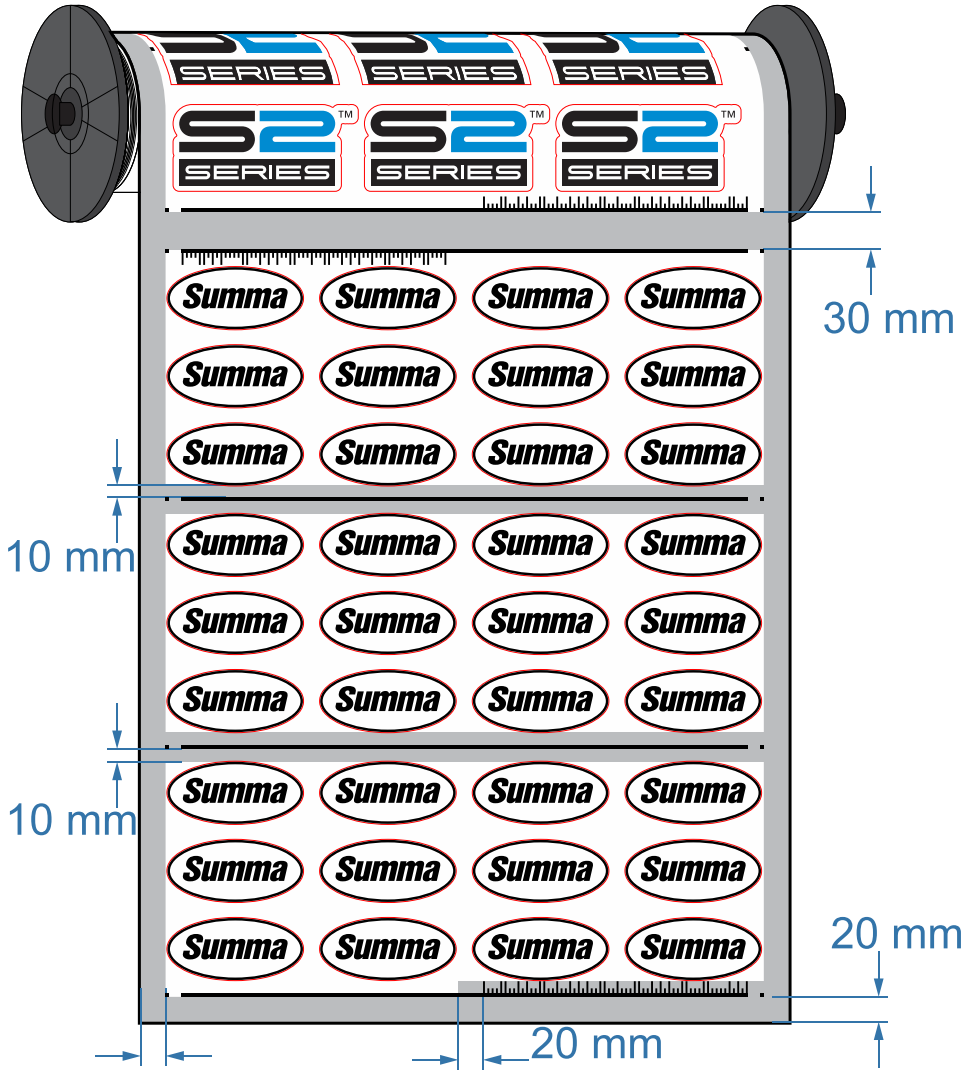
Illustration 4.10 - OPOS Barcode Size



Note: *In order not to mislead the OPOS sensor while looking for the job, the area outside the markers should remain blank. All extra's (job-references, color-bars, ..) should be placed inside the job area.*

In following figure the areas that need to stay blank are made gray.

Illustration 4.11 - OPOS BARCODE Margins



Note: *When using the cut-off option, then the distance between 2 jobs must be at least 30 mm more than the cut-off margin.*

The OPOS BARCODE workflow can be started from the software or from the cutter UI. To start it from the software a command is used similar to setting the alignment method.

```
[ESC];@:
SET SPECIAL_LOAD = OPOS_BARCODE.
LOAD_MARKERS.
END.
```

NOTE: *Use the "SET SPECIAL_LOAD = OPOS_BARCODE." in combination with "LOAD MARKERS." only to start the barcode workflow. Do not use this command in the cutfiles, as it is similar to the alignment mode command, the command interpreter of the cutter might misinterpret it as an alignment mode.*

4.5 Data sent to the cutter

4.5.1 OPOS Commands.

Before sending the cutting data (in DM/PL or HP-GL) to the cutter, the cutter must get information about the markers and the registration of the markers must be initiated. To do this the “encapsulated” language is used.

Following information must be sent to the cutter:

1. The size of the markers, both X-size and Y-size.
2. The distance between 2 markers in both directions.
3. The number of markers (in 1 row).
4. The type of registration procedure (in our case OPOS).
5. A command to start the registration of the markers.

This is done using the next encapsulated commands:

SET SPECIAL_LOAD = [OPOS,OPOS_XY,OPOS_XY2,OPOS_XTRA]

This command specifies which alignment method to use. In our case it will be OPOS.

SET MARKER_X_DIS = [a number in the range 1200 to 52000]

This command sets the marker X distance. The unit is 0.025 mm.

SET MARKER_Y_DIS = [a number in the range 1200 to 64000]

This command sets the marker Y distance. The unit is 0.025 mm.

SET MARKER_X_SIZE = [a number in the range 80 to 400]

This command sets the marker X Size. The unit is 0.025 mm.

SET MARKER_Y_SIZE = [a number in the range 80 to 400]

This command sets the marker Y Size. The unit is 0.025 mm.

SET MARKER_X_N = [a number in the range 2 to 128]

This command specifies the number of markers **in 1 row** along the X-axis. So it does not specifies the total number of markers but only half of them.

SET SPECIAL_LOAD = OPOS_BARCODE.

This command in combination with the following command start up the barcode workflow.

LOAD_MARKERS.

This command starts (initiates) the procedure to register the markers on the cutter. After this command is sent to the cutter, the cutter will ask the user to set the optical sensor near the first marker. Then all the markers will be sensed automatically.

4.5.2 Cutting Data

The outline of the design can be sent in either DM/PL or HP-GL to the cutter. Important is that the origin of the cutting data is situated at the same place as the origin marker and more precisely at the Lower Right corner of this marker. To be sure that the origin is correct you can include the first marker in the outline data.

4.5.3 Sample File

Now follows an example of a cutting file that matches a printed design where markers of 2mm by 2mm where used. The distance between the markers is 400 mm in the X-direction and 1200 mm in the Y-direction. There are a total of 3 markers in one row, or 6 markers in total.

```
<esc>;@:
SET SPECIAL_LOAD=OPOS.
SET MARKER_X_DIS=16000.
SET MARKER_Y_DIS=48000.
SET MARKER_X_SIZE=80.
SET MARKER_Y_SIZE=80.
SET MARKER_X_N=3.
LOAD_MARKERS.
END.
```

<contour cutting information in DMPL, HPGL or HP–GL/2>

4.6 Automating OPOS

4.6.1 Introduction

There are 4 cases supported to facilitate the use of several jobs that must be cut with OPOS. The main purpose to automate OPOS is to reduce the user intervention and time. Only for the first job the user will have to select the first marker. Then no more user intervention will be needed.

1. Identical jobs on a roll.
2. Different jobs on a roll.
3. Identical jobs on several sheets.
4. Using a barcode system

The scenarios are described in the following sections.

4.6.2 Identical jobs on a roll

Use the following procedure to cut several identical signs printed on a roll.

- Send the OPOS parameters that fit the sign as described in section 4.5.
- Tell how far 2 signs are from each other. For this purpose set the RECUT_OFFSET parameter. The RECUT_OFFSET parameter must be equal to the distance between the last marker on the first sign and the first marker on the next sign.
  Note: When using the cut-off option, then the distance between 2 jobs must be at least 30 mm more than the cut-off distance.
- Then send the LOAD_MARKERS command.
- Then send the contour data (DMPL). End this DMPL file with the DMPL ‘e’ command (end of plot command).

Sample1 shows how to automate small jobs (less than 8 MB)
Sample 2 shows how to automate bigger jobs.

SAMPLE 1:

```
<ESC>;@:
SET SPECIAL_LOAD=OPOS.
SET MARKER_X_SIZE 80.
SET MARKER_Y_SIZE 80.
SET MARKER_X_DIS 4400.
SET MARKER_Y_DIS 3920.
SET MARKER_X_N 2.
SET RECUT_OFFSET 40.
LOAD_MARKERS.
END.
;:ECN A U 2 2 D 1935 2 1935 1817 2 1817 2 2 U 1935 1000 e @

<ESC>;@:
RECUT 3
END.
```

SAMPLE 2:

```
<ESC>;@:
SET SPECIAL_LOAD=OPOS.
SET MARKER_X_SIZE 80.
SET MARKER_Y_SIZE 80.
SET MARKER_X_DIS 4400.
SET MARKER_Y_DIS 3920.
SET MARKER_X_N 2.
SET RECUT_OFFSET 40.
LOAD_MARKERS.
END.
;:ECN A U 2 2 D 1935 2 1935 1817 2 1817 2 2 U 1935 1000 e @
```

4.6.3 Different jobs on a roll

Use the following procedure to cut several different signs printed on a roll.

1. Send the OPOS parameters that fit the first sign as described in section 4.5.
2. Send the “LOAD_MARKERS” command.
3. Then send the contour data of the first sign (DMPL). DO NOT End this DMPL file with the DMPL ‘e’ command (end of plot command)
4. Now use the SET_ORIGIN command to tell where the first marker of the next sign is compared to your current position.
5. Send the OPOS parameters that fit the second sign.
6. Then send the “LOAD_MARKERS” command.
7. Then send the contour data of the second sign (DMPL). DO NOT End this DMPL file with the DMPL ‘e’ command (end of plot command)
8. Repeat step 4 to 7 as much as necessary.

SAMPLE 3:

```
ESC;@:
SET SPECIAL_LOAD=OPOS.
SET MARKER_X_SIZE 120.
SET MARKER_Y_SIZE 120.
SET MARKER_X_DIS 3200.
SET MARKER_Y_DIS 3200.
SET MARKER_X_N 2.
LOAD_MARKERS.
END.
;:ECN A U 2 2 D 1935 2 1935 1817 2 1817 2 2 U 1935 0 0,0 @.
```

4.6.4 Identical jobs on several sheets

Use the following procedure to cut several identical signs printed on several sheets.

1. Send the OPOS parameters that fit the sign as described in section 4.5.
2. Turn on the OPOS_SHEET_MODE.
3. Send the “LOAD_MARKERS” command.
4. Send the contour data (DMPL).
5. Now the user will have to remove the sheet of media and when he inserts a new sheet, the markers will be sensed and the same sign will be cut again. This will occur until the user presses RESET on the front panel.

SAMPLE 4

```
<ESC>;@:
SET SPECIAL_LOAD=OPOS.
SET MARKER_X_SIZE 120.
SET MARKER_Y_SIZE 120.
SET MARKER_X_DIS 3200.
SET MARKER_Y_DIS 3200.
SET MARKER_X_N 2.
SET OPOS_SHEET_MODE ON.
LOAD_MARKERS.
END.
;:ECN A U 2 2 D 1935 2 1935 1817 2 1817 2 2 U 1935 0 @.
```

4.6.5 OPOS BARCODE

The barcode system is used when several jobs are printed after each other. Each job has a barcode that indicates the jobnumber.

After activation, OPOS will start looking for the first job and scan the barcode. The jobnumber will be sent to the computer. On the computer a kind of **'Barcode server'** should be running, monitoring the port for incoming jobnumbers. On receipt of the jobnumber the software should select the correct contour job and sent the data as a standard OPOS job. OPOS will start scanning the job (with the OPOS information it received in the file) and cut the contours.

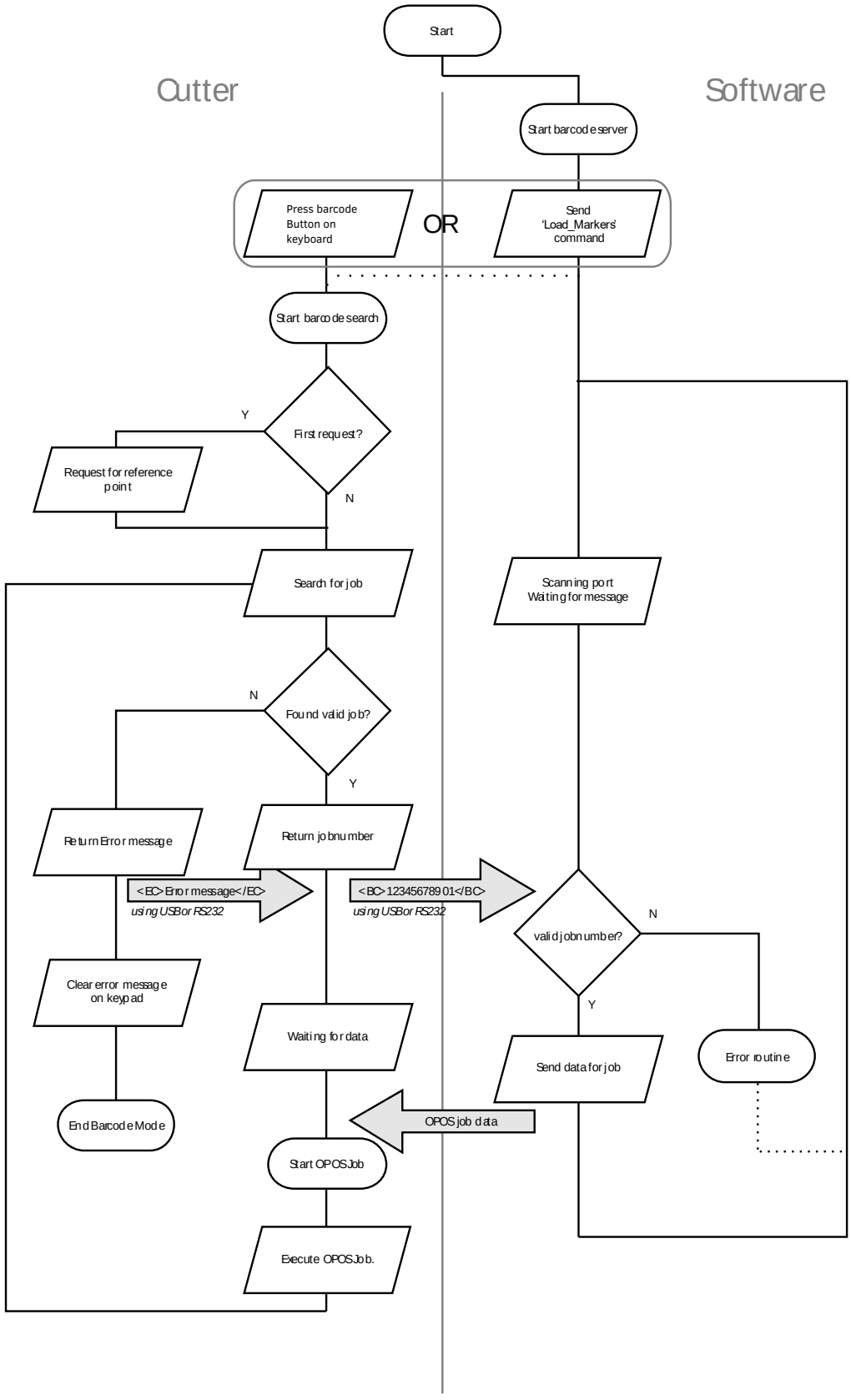
Note: *While the barcode server on the computer is running and the computer is waiting for a new jobnumber, the communication port should be kept open in order not to lose connection with the cutter.*

When the job is finished, it will automatically start looking for the next job. When it finds a new job with valid barcode it will sent the jobnumber to the computer and wait for the data.

This continues until no new valid job is found or if the computer doesn't return any data.

Barcode server in Summa Cutter Control:
Summa Cutter Control is a Windows based utility to monitor all parameters of the cutter from the computer. In version 4.8 (or later) a barcode server is implemented. This can be used for testing during implementation. But can also be used as a final solution in combination with other software. For the barcode server, the OPOS jobs need to be stored in ready to sent data files. File name should be the barcode number and the extension can be *.plt *.dmpl *.dmp *.hpgl *.hpg or *.prn (eg: *12345678901.plt*)

Illustration 4.12 - OPOS Barcode workflow



Use the following procedure to activate the barcode system.

1. Send the “SPECIAL_LOAD = OPOS_BARCODE” command.
2. Send the “LOAD_MARKERS” command.
   Note: these first two steps can also so be executed on the keyboard of the Summa Cutter.
3. Wait until a jobnumber (11-digit number) is receipt.
   Returnformat: <BC>###########</BC>
4. Send the OPOS parameters that fit the jobnumber as described in section 4.5.
5. Send the “LOAD_MARKERS” command.
6. Then send the contour data of the first sign (DMPL). End this DMPL file with the DMPL **‘@’** command (deselect command).

- After cutting the contour and receipt of the ‘@’ command, OPOS will start looking a next job and return the next jobnumber. Repeat step 3 to 6 as much as necessary.
- If no new job is found or the barcode is invalid a error string is returned: <EC>error message</EC>.

SAMPLE:

```
<ESC>;@:
SET SPECIAL_LOAD = OPOS_BARCODE
LOAD_MARKERS.
END.
```

Wait for job number

Returned: <BC>12345678901</BC>

```
<ESC>;@:
SET SPECIAL_LOAD OPOS_XY
SET MARKER_X_SIZE 80.
SET MARKER_Y_SIZE 80.
SET MARKER_X_DIS 4400.
SET MARKER_Y_DIS 3920.
SET MARKER_X_N 2.
LOAD_MARKERS.
END.
;:ECN A U 2 2 D 1935 2 1935 1817 2 1817 2 2 U 1935 1000 @
```

Wait for job number

Returned: <BC>10987654321</BC>

```
<ESC>;@:
SET SPECIAL_LOAD OPOS
SET MARKER_X_SIZE 80.
SET MARKER_Y_SIZE 80.
SET MARKER_X_DIS 4500.
SET MARKER_Y_DIS 3960.
SET MARKER_X_N 3.
LOAD_MARKERS.
END.
;:ECN A U 2 2 D 1935 2 1935 1817 2 1817 2 2 U 1935 1000 @
```

Wait for job number ...

ERROR MESSAGES:

```
<EC>error message</EC>
```

If OPOS doesn't find a new job (=horizontal line) it will return following error message:

```
<EC>Unable to sense OPOS XY correction line.</EC>
```

All these messages will be followed by a general OPOS Barcode error:

```
<EC>Unable to sense OPOS Barcode.</EC>
```

e.g: if the checksum is incorrect following is returned:

```
<EC>Checksum failed!</EC><EC>Unable to sense OPOS Barcode.</EC>
```

Other OPOS messages are:

```
<EC>Media not loaded!</EC>
<EC>Loading OPOS cancelled.</EC>
<EC>At least 2 markers must be sensed.</EC>
<EC>Unable to sense OPOS XY correction line.</EC>
<EC>The origin is not defined correctly!</EC>
```

5 Cutting Through.

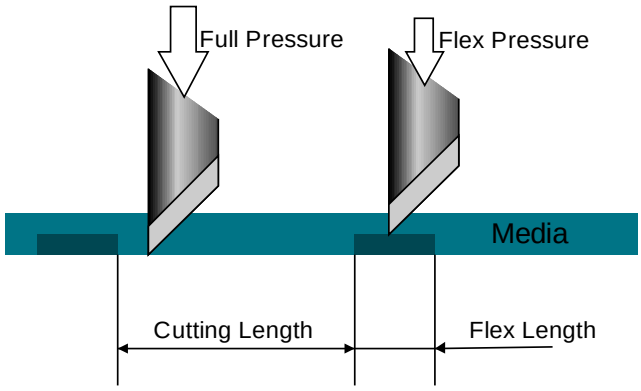
5.1 Introduction.

Cutting material completely through on a drum-plotter is not an easy thing to do. It should be avoided that cut pieces fall out while proceeding the job because this causes material crashes. In order to avoid these crashes Summa implemented 'FlexCut' more than 10 years ago. An interrupted cutting line makes sure that the material remains together thanks to the small media 'bridges'. When the job is finished the cut pieces can be torn out.
Although FlexCut helps a lot, it still has several limitations. Often, it is difficult to find the correct balance between cutting deep enough making sure the pieces can be taken out easily, and not cutting too deep making sure the material keeps it strength while cutting. Sometimes this balance doesn't exist meaning that this material can't be cut.
On the latest Summa models new functionality has been implemented. A single tool-command activates FlexCut, panelling and intelligent vector optimization routines. This way more materials can be cut-through reliably.

5.2 FlexCut

When FlexCut is activated on the Summa cutter. The cutting line will become an interrupted line. The different lengths and pressures of the cut line can be set with encapsulated commands. Activating FlexCut can be done with encapsulated commands but doing this is normally not done (see important note below).

Illustration 5.1 - FlexCut



Note: *It is not recommended to use encapsulated commands to activate FlexCut. Use instead the tool 6 and 10 as explained in next paragraph, the use of tools activates extra useful features. Also encapsulated commands in the middle of cut data mess up other cutter functions like recut and panelling.*

5.3 Tool 6 & Tool 10

When activating tool 6 or tool 10, the S Class 3 cutter will combine the FlexCut with panelling and optimization routines. Tool 10 uses only the automatic panelling, not the extra optimization routines.

The flex panel size can be set with encapsulated commands (see 3.1.22 FLEX_PANEL_SIZE = [a number in the range 1 to 250]).

It is important that all objects that will be cut with this tool are **grouped together at the end** of the cutting data.

Only lines to cut through:

```
;:ECN A P6 U 2 2 D 1935 2 1935 1817 2 1817 2 2 U 1935 1000 e
```

First using default settings and then cutting through:

```
;:ECN A U 102 102 D 1835 102 1835 1717 102 1717 102 102 P6 U 2 2 D 1935 2 1935
1817 2 1817 2 2  U 1935 1000 e
```

In combination with OPOS:

```
<ESC>;@:
SET SPECIAL_LOAD OPOS
SET MARKER_X_SIZE 80.
SET MARKER_Y_SIZE 80.
SET MARKER_X_DIS 4400.
SET MARKER_Y_DIS 3920.
SET MARKER_X_N 2.
LOAD_MARKERS.
END.
;:ECN A U 102 102 D 1835 102 1835 1717 102 1717 102 102 P6 U 2 2 D 1935 2 1935
1817 2 1817 2 2  U 1935 1000 e
```

5.4 Guidelines

The settings for cutting through depend mainly on the material but also on the type of cutting head (drag / tangential) and the wear of the knife. In order to cut through a standard vinyl following settings should be a good start (feel free to use other settings).

Cutting Length : 10mm
Flex Length : 1 mm
Full pressure: 250gr

In order to avoid high knife wear and damage to the cutting strip, full pressure may not be too high. Machine operator should also make sure the knife depth is set accordingly.

Flex pressure: 80-130gr
Vary this parameter in order to get an acceptable FlexCut line.

Flex panel size: 5-10cm (on more difficult materials like paper it may be recommend to reduce the flex panel size to 2cm)

```
<ESC>;@:
SET FLEX_PANEL_SIZE = 5.
END.
;:ECN A P6 U 2 2 D 1935 2 1935 1817 2 1817 2 2 U 1935 1000 e
```

Note: *Be careful when sending parameters to the cutter with every job. In some cases the operator may have calibrated the cutter correctly not knowing that the software will overrule his settings! Summa recommends that the operator can choose in the software if parameters are sent or not.*

The cutting through functionality is focused to cut convex polygons. However it will process any data it receives in this mode. But it is obvious that the more complex the shape is, the more difficult it will be to get an acceptable result.
When cutting through, it is recommended that parallel lines are at least 1 cm away from each other. Otherwise, while cutting the second line, the first line may come loose and cause trouble.
In case the cutter is not capable of cutting the material in one pass. It is possible to cut each cutline several times (see 3.1.24 MULTIPASS = [a number in the range 1 to 7]).

Note: *Do not send encapsulated commands once actual cutdata has been send Doing so interferes with features like recut, panelling and FlexCut.*

6 TCP/IP

6.1 Introduction

In order to talk to a Summa cutter through Ethernet of WiFi you need to implement a TCP/IP protocol. Besides the IP address, you also need to write or read to/from a certain port. For the S Class 3 cutter this is set fixed to **port 9100.** The IP address can be set to any IP address.

Recommendations when using TCP/IP communication
When polling for the media size, build in a time-out of 10 seconds, it may take a couple of seconds before the cutter reacts with the media size.

On some operating systems care must be taken when closing a socket. Closing the socket too soon may result in data not being sent to the cutter. e.g. windows will not send the last chunk of data for large files. Windows calls this 'closing a socket gracefully'. Example code below shows how to implement a graceful close in socket communication for windows applications.

6.2 Description by MSDN on graceful close.

To assure that all data is sent and received on a connected socket before it is closed, an application should use shutdown to close connection before calling closesocket. One method to wait for notification that the remote end has sent all its data and initiated a graceful disconnect uses the WSAEventSelect function as follows :

1. *Call WSAEventSelect to register for FD_CLOSE notification.*
2. *Call shutdown with how=SD_SEND.*
3. *When FD_CLOSE received, call the recv or WSARecv until the function completes with success and indicates that zero bytes were received. If SOCKET_ERROR is returned, then the graceful disconnect is not possible.*
4. *Call closesocket.*

If this cannot be implemented then a trick can be used to prevent losing the last chunk of data, by adding a delay of 3 - 4 seconds before closing the socket. No guaranty can be given that all data will have been sent to the cutter.
Also do not close and open the socket all the time, leave it opened until all data is sent.
Recommended implementation
Through this example code, the following general handle is used for the client socket.

```
SOCKET ConnectSocket = INVALID_SOCKET;
```

You have to include following header to support sockets.

```
#include <Winsock2.h>
```

Your project should be linked to the ws2_32.lib to support the code shown below.

```
#pragma comment(lib, "ws2_32.lib")
```

Opening a socket

```
int iResult;
WSADATA wsaData;

struct sockaddr_in clientService;
```

```
// Initialize Winsock
iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
if (iResult != NO_ERROR) {
//        wprintf(L"WSAStartup failed with error: %d\n", iResult);
ConnectSocket = INVALID_SOCKET;
return;
}

// Create a SOCKET for connecting to server
ConnectSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (ConnectSocket == INVALID_SOCKET) {
//        wprintf(L"socket failed with error: %ld\n", WSAGetLastError());
          WSACleanup();
          ConnectSocket = INVALID_SOCKET;
          return;
}

// The sockaddr_in structure specifies the address family, IP address, and port of
clientService.sin_family = AF_INET;
clientService.sin_addr.s_addr = inet_addr( AnsiString(this->edHost->Text).c_str() );
clientService.sin_port = htons( this->edPort->Text.ToInt() );

// Connect to server.
iResult = connect( ConnectSocket, (SOCKADDR*) &clientService, sizeof(clientService) );
if (iResult == SOCKET_ERROR) {
//        wprintf(L"connect failed with error: %d\n", WSAGetLastError() );
closesocket(ConnectSocket);
WSACleanup();
ConnectSocket = INVALID_SOCKET;
return;
}
```

Put socket in Blocking-mode

TCP/IP sockets are default set to blocking mode. To be sure you can use the code below to setup sockets in blocking-mode.

```
// Set the socket I/O mode: In this case FIONBIO enables or disables the blocking mode for the socket
// If iMode = 0, blocking is enabled;
// If iMode != 0, non-blocking mode is enabled.
iMode = 0;
iResult = ioctlsocket(ConnectSocket, FIONBIO, &iMode);
if (iResult == SOCKET_ERROR) {
//        printf("ioctlsocket failed with error: %ld\n", iResult);
}
```

Set send timeout

When using sockets in blocking mode, you should set a timeout for sending data, to avoid the main program from freezing.

```
// Set send timeout.
iOptVal = 10000;
iResult = setsockopt( ConnectSocket, SOL_SOCKET, SO_SNDTIMEO, (char *) &iOptVal, iOptLen );
if (iResult == SOCKET_ERROR) {
//      wprintf(L"setsockopt for SO_SNDTIMEO failed with error: %u\n", WSAGetLastError());
        closesocket(ConnectSocket);
        WSACleanup();
        ConnectSocket = INVALID_SOCKET;
        return;
}
```

For sending encapsulated data, this is no problem.
But for the cutting data you should keep sending data, for example when the Cutter is offline, and not processing any data anymore. You can achieve this in 2 ways:

- Send the file in a different thread than the main thread and setting iOptVal = 0.
- Send the file in the main thread, setting iOptVal = 10000, and retry each time. Decrease the delay to make you main program more responsive.

Set receive timeout

When using sockets in blocking mode, you should set a timeout for receiving data, to avoid the main program from freezing.

```
// Set receive timeout.
iOptVal = 500;
iResult = setsockopt( ConnectSocket, SOL_SOCKET, SO_RCVTIMEO, (char *) &iOptVal, iOptLen );
if (iResult == SOCKET_ERROR) {
//      wprintf(L"setsockopt for SO_SNDTIMEO failed with error: %u\n", WSAGetLastError());
        closesocket(ConnectSocket);
        WSACleanup();
        ConnectSocket = INVALID_SOCKET;
        return;
}
```

Closing the socket

```
int iResult;
WSAEVENT NewEvent;

long events;
WSANETWORKEVENTS wsaevents;


// shutdown the connection since no more data will be sent
iResult = shutdown(ConnectSocket, SD_SEND);
if (iResult == SOCKET_ERROR) {
//      wprintf(L"close failed with error: %d\n", WSAGetLastError());
        WSACleanup();
        ConnectSocket = INVALID_SOCKET;
        return;
}

// wait for other side to close the socket
NewEvent = WSACreateEvent();
WSAEventSelect( ConnectSocket, NewEvent, FD_CLOSE);
do
{
        WaitForSingleObject(NewEvent, 50);
        WSAEnumNetworkEvents(ConnectSocket, NewEvent, &wsaevents);
        events = wsaevents.lNetworkEvents;
} while ( !(events & FD_CLOSE) );

// cleanup
iResult = closesocket(ConnectSocket);
if (iResult == SOCKET_ERROR) {
//      wprintf(L"close failed with error: %d\n", WSAGetLastError());
        WSACleanup();
        ConnectSocket = INVALID_SOCKET;
        return;
}

WSACleanup();
ConnectSocket = INVALID_SOCKET;
```

Sending data

When send succeeds it returns the number of bytes it has sent. To send files we split up the data in packets of 1024 bytes.

```
iResult = send( ConnectSocket, buffer,sizeof(buffer), 0 );
if (iResult == SOCKET_ERROR) {
//      wprintf(L"send failed with error: %d\n", WSAGetLastError());
        WSACleanup();
        ConnectSocket = INVALID_SOCKET;
        return;
}
```

Receiving data

```
iResult = recv(ConnectSocket, ReadBuffer, 1500, 0);
if (iResult == SOCKET_ERROR) {
//      wprintf(L"recv failed with error: %d\n", WSAGetLastError());
        WSACleanup();
        ConnectSocket = INVALID_SOCKET;
        return;
}
```

6.3 Discovering Ethernet devices

The S Class 3 cutter support discovery on a local network using broadcast messages. This is again done by using the Winsock2 library.
Broadcasting must be done through the UDP protocol, so you will need to create an UDP socket.

- Message to send: PingForSumma
- Received message:
  MODEL_NAME;SERIAL NUMBER;IP ADRES
  e.g.S3T75;T12302-10001;246.124.250.100
- Port to use: 9000

CODE SAMPLE:

Using Winsock2

```
#include <Winsock2.h>
#include <string> // for using std::string
#pragma comment(lib, "ws2_32.lib")
```

Setting the message to send

```
std::string sMsg = “PingForSummaS2”;
```

Setting the ethernet port

```
USHORT  ushPort = 9000;
```

Creating and opening the socket for UDP broadcasting

```
int iResult;
WSADATA wsaData;
SOCKET ConnectSocket = INVALID_SOCKET;

struct sockaddr_in clientService;

// Initialize Winsock
iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
if (iResult != NO_ERROR)
{
  ConnectSocket = INVALID_SOCKET;
  return;
}

// Create a SOCKET for connecting to broadcast
```

```
ConnectSocket = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (ConnectSocket == INVALID_SOCKET)
{
  WSACleanup();
  ConnectSocket = INVALID_SOCKET;
     return;
}

// The sockaddr_in structure specifies the address family,
// IP address, and port
clientService.sin_family = AF_INET;
clientService.sin_addr.s_addr = htonl(INADDR_ANY);
clientService.sin_port = htons(ushPort);

// Bind the socket to any address and the port
iResult = bind(ConnectSocket, (SOCKADDR*) &sockAddr, sizeof(sockAddr));
if (iResult != 0)
{
  WSACleanup();
  ConnectSocket = INVALID_SOCKET;
     return;
}

// Enable broadcast
bool bOptVal = true;
iResult = setsockopt(ConnectSocket, SOL_SOCKET, SO_BROADCAST, (char *) &bOptVal, sizeof
(bOptVal) );
if (iResult == SOCKET_ERROR)
{
  WSACleanup();
  ConnectSocket = INVALID_SOCKET;
     return;
}

// Set receive timeout (here set to 600ms)
iOptVal = 600;
iResult = setsockopt(ConnectSocket, SOL_SOCKET, SO_RCVTIMEO, (char *) &iOptVal, sizeof(iOptVal) );
if (iResult == SOCKET_ERROR)
{
  WSACleanup();
  ConnectSocket = INVALID_SOCKET;
  return;
}
```

Sending the broadcast message

```
// Local variables
int iResult;
const int BufLen = 512;
int iLastErr;
```

```
sockaddr_in raddr;
int rlen = sizeof (raddr);

// Clear ping info
this->vInfo.clear();

// The sockaddr_in structure specifies the address family,
// IP address, and port of the target socket
sockaddr_in        sockAddr;
sockAddr.sin_family = AF_INET;
sockAddr.sin_addr.s_addr = htonl(INADDR_BROADCAST);
sockAddr.sin_port = htons( ushPort);

  // Send ping command
iResult = sendto(ConnectSocket, sMsg.c_str(),sMsg.size(), 0, (SOCKADDR *) &sockAddr,
sizeof(sockAddr));
if (iResult == SOCKET_ERROR)
{
  WSACleanup();
  ConnectSocket = INVALID_SOCKET;
  return;
}
```

Receiving data

```
// Read buffer
char ReadBuffer[512];

// Wait time
const clock_t MAX_PING_WAIT_TIME = 600;

// Read data
clock_t starttime = clock();
clock_t time;
do
{
  iResult = recvfrom(ConnectSocket, ReadBuffer, BufLen, 0, (SOCKADDR*)&raddr, &rlen);
  if( iResult == SOCKET_ERROR )
  {
          iLastErr = WSAGetLastError();
          if( iLastErr != WSAETIMEDOUT )
          {
                  WSACleanup();
                  ConnectSocket = INVALID_SOCKET;
                          return;
          }

          // Add NULL char to terminate data in read buffer for Summacut
          if( iResult > 0 )
                  ReadBuffer[iResult] = '\0';
```

```
            // AT THIS POINT, YOU SHOULD PARSE THE DATA IN READBUFFER AS DESCRIBED BEFORE
        }

         Sleep(10);

        time = clock() - starttime;
    }
    while( time < MAX_PING_WAIT_TIME );
```

Disconnecting the socket

```
ConnectSocket = INVALID_SOCKET
WSACleanUp();
```

7 USB

7.1 Introduction

This document describes the basics to communicate with the USB bus on the cutters from Summa, this for the windows operating system.
First some basics about the USB bus and the support of windows for this bus are described.
Then the specific details of the software-architecture for communication with the cutters are described.
The document ends with a sample written in C that clarifies the theory.

Note: *The explanation is done for 32 bit applications, For 64 bit applications it is completely the same providing you use the 64 bit DLL and drivers.*

7.2 USB and Windows

USB is a standard bus available on all PC's. This bus offers serial communication at high speed. It is hot pluggable, which means devices can be attached or removed at any time from the PC.

The Win32 API (or let say Windows) offers some functions to communicate. These functions are `CreateFile()`, `ReadFile()`, `WriteFile()` and `CloseHandle()`.
More info about these functions can be found in your development tools (Visual C++ or Borland C++ or MSDN). The term 'file' also includes communications resources such as the USB port.
The `CreateFile()` is used to get a handle to a file or communication resource. The `WriteFile()` is then used to send data to the opened 'file'. The `CreateFile()` function can **NOT** be used directly with the USB bus. Instead a function (`open_file()`) provided by Summa must be used. This function does the same as `CreateFile()`; it returns a handle to the USB bus for the cutter. Then the `WriteFile()` and `ReadFile()` functions can be used.

7.2.1 USB Pipes

Each USB device (in our case the cutter) communicates to the host software (= software on PC) through what is called pipes. Most USB devices do have several pipes implemented. The cutter has 3 pipes implemented. This means that there are 3 communications channels between the cutter and the host software.

The two most important pipes are `PIPE00` and `PIPE01`. `PIPE01` is an **unidirectional** pipe that transports data from the host PC to the cutter. This data is the cut data (DM/PL or HP-GL). `PIPE00` is also **unidirectional** and carries information from the cutter to the host. For example you can get the size of the media through this pipe (in DM/PL for example).

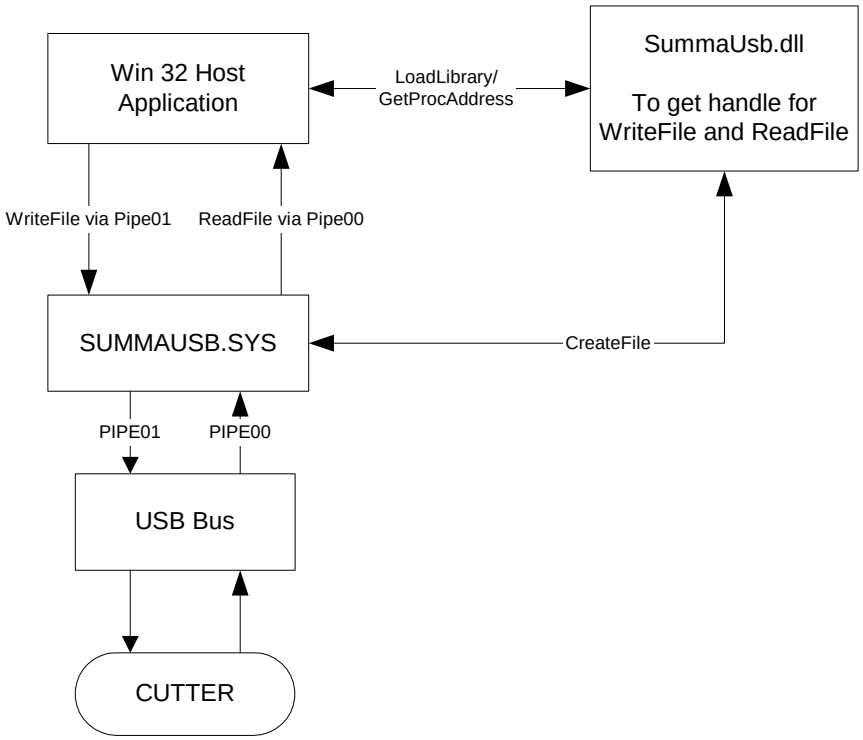
Sending data has to be handled over `PIPE01` and receiving data will be handled over `PIPE00`. Each pipe will need a different handle.
As an example to get the media size of the cutter (in DM/PL) you will have to send *`;: ER`* on `PIPE01` and you will have to read the response on `PIPE00`.

7.3 Win32 Software Components

Summa has written a windows driver (SummaUsb.sys) that allows Windows programs to communicate with the cutter over the USB bus with the standard Win32 API functions `ReadFile()`, `WriteFile()` and `CloseHandle()`.
Before using this functions, you must get a handle to the USB cutter device. For this use '`open_file()`', a function that Summa has written. This function is made available through the SummaUsb.dll. (see the sample in section 7.5 on how to use the DLL).

You must use the `WriteFile()` and `ReadFile()` functions to communicate over the USB bus.

Illustration 7.1 - The different USB software parts



7.3.1 SummaUsb.sys

The Plug and play Manager from windows automatically load this WDM driver when it detects that a cutter has been plugged into the USB bus. This driver a be downloaded from the Summa website.

> ***Note:*** *The USB bus allows hot plugging and unplugging of the devices attached to its bus. The cutter may be plugged in after your program has started. You will not get a handle when the cutter is not plugged in or is powered off. You will only get a handle to the device after the cutter is powered on and plugged in the USB bus.*

Get a handle to the USB device just before sending/receiving data. It is preferred not to get a handle at initialization of your program.

7.3.2 SummaUsb.dll

SummaUsb.dll is a dynamic link library that helps you to get an easy access to the handles of the USB pipes from the cutter. You can get 2 handles to the cutter, one for each pipe. After you have a handle to the USB cutter, you don't need SummaUsb.dll anymore. You can use the standard win32 API functions `ReadFile()` and `WriteFile()` to communicate with the cutter.

You will have to include this file with the installation of your product!

This DLL has 4 functions called `open_file()`, `open_file2()`, `open_file3()` and `open_file4()` that does the same as `CreateFile()`:
One function for each of the 4 available USB ports. The USB port number can be set through the control panel of the cutter.

HANDLE **__stdcall** open_file (**char*** lpFileName, **DWORD** *dwFlagsAndAttributes*);
HANDLE **__stdcall** open_file2 (**char*** lpFileName, **DWORD** *dwFlagsAndAttributes*);
HANDLE **__stdcall** open_file3 (**char*** lpFileName, **DWORD** *dwFlagsAndAttributes*);
HANDLE **__stdcall** open_file4 (**char*** lpFileName, **DWORD** *dwFlagsAndAttributes*);

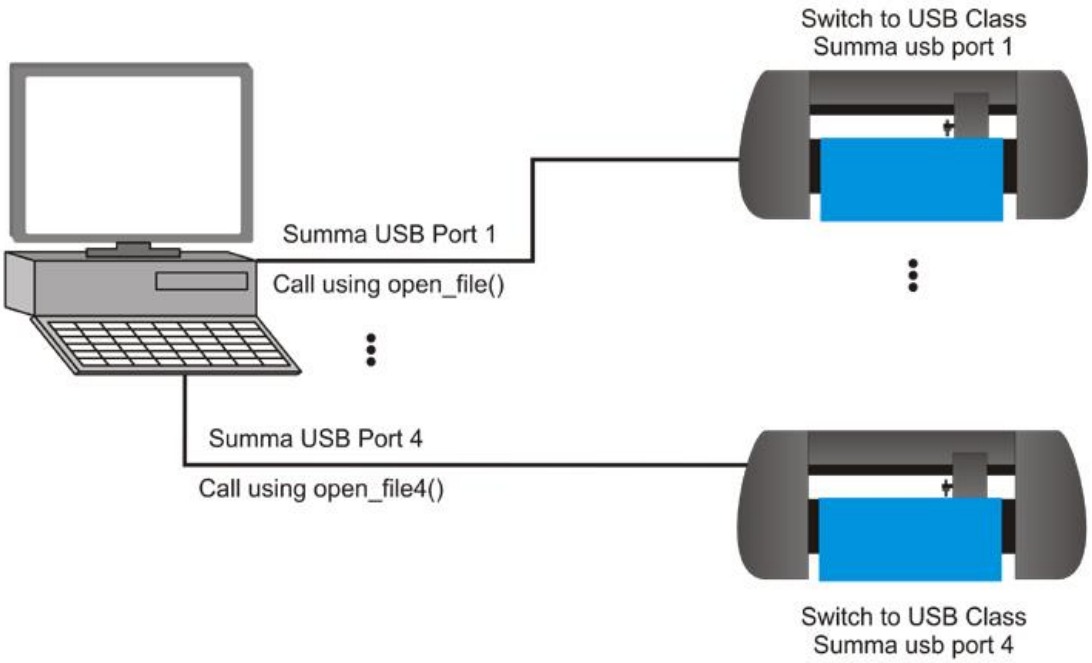
Parameters

lpFileName
Points to a string that specifies the name of the pipe. The only valid values for that parameter are the strings "`PIPE00`" and "`PIPE01`". "`PIPE00`" to read data from the cutter and "`PIPE01`" to write data to the cutter.

dwFlagsAndAttributes
Instructs the system whether or not to use asynchronous communication. See information about `CreateFile()`.

Values: `0 (NULL)` for synchronous operation or `FILE_FLAG_OVERLAPPED` for asynchronous operation.

Illustration 7.2 - Using multiple USB cutter devices



Return Values

If the function succeeds, the return value is an open handle to the specified pipe. If the function fails, the return value is `INVALID_HANDLE_VALUE`. To get extended error information, call `GetLastError()`. The pipe cannot be opened when the cutter is powered off or when the cutter is not connected to the USB bus.

Note: *This function must be declared as* `__stdcall`.

`WriteFile()` and `ReadFile()` can then use the handle returned by `open_file()`.

7.3.3 Win32 Host Application

There are some restrictions on the Win32 API functions `ReadFile()` and `WriteFile()`.

Writing Data To Cutter.

You must send the data in little packets (e.g. 256 bytes). This gives much better performance. When sending a file of 1 MB, for instance, the USB driver stack will take up too much time and your computer will seem to hang. When sending little chunks it is also easier to cancel a current job being sent.

When the data is split in packets of 256 bytes, the last packet will probably be smaller than 256 bytes, this is no problem.

If the input buffer of the cutter is full, the `WriteFile()` function will not return, but it will wait until some place in the buffer is available.

Reading Data From Cutter.

The timeout principle doesn't exists on the USB bus, at least not in the same way as in the serial port. So when querying info from the cutter, you normally first send data on `PIPE01` and then read data on `PIPE00`. The cutter needs some time to put data on the USB bus after receiving the request. While the data is not ready the cutter will respond to any query with a zero length packet. This means the `ReadFile()` function will return with 0 bytes read. You will have to call `ReadFile()` until you get the desired response from the cutter. Calling `ReadFile()` only once will not be enough, the cutter needs in most cases to have 2 `ReadFile()` commands, the first one will respond with 0 bytes, the second one with the desired response (if the cutter is ready to answer).

So when getting an answer from the cutter of zero bytes or less than expected, it does not mean that no data is available anymore. The cutter needs some time to put the data in its output buffer. It is recommended to keep reading until the expected answer is received.

Reading data must be done by multiples of 16. Do not read 1 byte at a time with the `ReadFile()` function! When reading more data than available, the `ReadFile()` will return with success, but the number of bytes read will be set to what is really read.

7.4 Summary

- Make a 32-bit program (Win32) using `ReadFile()` and `WriteFile()` to communicate.
- Load the DLL SummaUsb.dll using `LoadLibrary()`.
- Get the address of the function `open_file()` using `GetProcAddress()`.
- Use `open_file()` from SummaUsb.dll to get a handle to the USB port instead of `CreateFile()`.
- Open 2 handles, one for reading data and one for writing data.
- Send data in little chunks (e.g. 256 bytes) with `WriteFile()`.
- Read data in multiples of 16 bytes with `ReadFile()`.
- When reading data the cutter may respond with 0 bytes. Make sure to keep reading until the expected answer is received before deciding the cutter doesn't respond.
- USB has no time-out function.

7.5 Win32 Sample Application

The sample is compiled using Visual C++ 5.0. The sample is a win 32 console application (runs in a 'DOS' box). Files are distributed together with this document. The RWSumma.c file gives an example on how to open handles to the USB pipes and how to send and read data. The sample also shows how to use SummaUSb.dll and its function "`open_file()`".
The method used in this sample is the so-called Run-Time Dynamic Linking. It has the advantage that the process can continue running even if the DLL is not available. The program can then notify the user of an error. If the user can provide the full path of the missing DLL, the process can use this information to load the DLL even though it is not in the normal search path.
This sample does not allow to cancel data or stop the execution of the `ReadFile()` or `WriteFile()` function. It has no time-out function implemented.

```
/*++
Copyright (c) 1999  Summa N.V.
Module Name:
 RWSumma.c
```

```
Abstract:
 Console test app for SummaUsb.sys driver
Environment:
 user mode only
Notes:
     Copyright (c) 1999 Summa N.V.  All Rights Reserved.

Revision History:
 11/11/99: created
--*/

#include <windows.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>
#include <sys\timeb.h>
#include <basetyps.h>

char inPipe[32] = "PIPE00";    // pipe name for bulk input pipe on our test
board
char outPipe[32] = "PIPE01";   // pipe name for bulk output pipe on our test
board
int gDebugLevel = 1;       // higher == more verbose, default is 1, 0 turns
off all
int WriteLen = 0;          // #bytes to write
int ReadLen = 0;           // #bytes to read

// prototype for open_file function from summausb.dll
typedef HANDLE (__stdcall *OPENFILE)(char*, DWORD);
OPENFILE POpenFile;

// functions
/*++
Routine Description:
 waits for a time ms (in milliseconds)
Arguments:
 ms : time to wait in milliseconds
Return Value:
 Zero
--*/
void delayms(double ms)
{
 double elapsed_time = 0;
 struct _timeb start, finish;

 _ftime( &start );
 _ftime( &finish );

 do
 {
       _ftime( &finish );
       elapsed_time = ((finish.time - start.time) * 1000) + finish.millitm
                                                          - start.millitm;
 } while (elapsed_time < ms);
}

int _cdecl main(
```

```
 int argc,
 char *argv[])
/*++
Routine Description:
 Entry point to RWsumma.exe
 Sends data to the USB cutter and reads response.
Arguments:
 argc, argv  standard console  'c' app arguments
Return Value:
 Zero
--*/
{
 char *pinBuf = NULL, *poutBuf = NULL;
 int nBytesRead, nBytesWrite,TotalBytesRead;
 ULONG i;
 UINT success;
 HANDLE hRead = INVALID_HANDLE_VALUE, hWrite = INVALID_HANDLE_VALUE;
 ULONG totalBytes = 0L;
 char DebugString2[] = " \x1B;@:.MENU. ";
 char DebugString[] = ";: EC1 ER ";

 UINT bResult = 0;

 HINSTANCE hinstLib;
 BOOL fRunTimeLinkSuccess = FALSE;

 // First the SummaUsb Dll will be loaded
 // then we will get the adress of the function needed in the dll

 // Get a handle to the DLL module.
 hinstLib = LoadLibrary("summausb.dll");

 // If the handle is valid, try to get the function address.
 if (hinstLib != NULL)
 {
       POpenFile = (OPENFILE) GetProcAddress(hinstLib, "open_file");

       // If the function address is invalid, print a error message

       fRunTimeLinkSuccess = (POpenFile != NULL);
       // If unable to call the DLL function, DLL must be corrupted?
       if (! fRunTimeLinkSuccess)
             printf("Error : could not open function OpenFile");
 }
 else
 {
       printf("Error : could not open Summausb.dll\n");
       return (1);
 }

 //
 // open the Read and Write Pipes
 //

 hWrite = (POpenFile)( outPipe, (DWORD)NULL);

 if (hWrite == INVALID_HANDLE_VALUE)
 {
       printf("could not open %s", outPipe);
```

```
        return 0;
    }

    hRead = (POpenFile)(inPipe, (DWORD)NULL);

    if (hRead == INVALID_HANDLE_VALUE)
    {
        printf("could not open %s", inPipe);
        return 0;
    }

    //
    // put some data in the output buffer
    //

    WriteLen = sizeof(DebugString);
    poutBuf = malloc(WriteLen);
    sprintf(poutBuf, DebugString);

    // allocates some memory to put read data.
    ReadLen = 64;
    pinBuf = malloc(ReadLen + 1);

    if ( poutBuf && hWrite != INVALID_HANDLE_VALUE)
    {
        // skip any data that is still left in the buffer of the cutter
        // by reading data from the cutter, until no more data is available

        do
        {
            success = ReadFile(hRead, pinBuf, ReadLen, &nBytesRead, NULL);
            delayms(10);
            /* add some kind of timeout or abort procedure ? */
        } while (nBytesRead);

         //
         // send data to the cutter
        //
        WriteFile(hWrite, poutBuf, WriteLen, &nBytesWrite, NULL);
         printf("<%s> W (%04.4d) : request %06.6d bytes -- %06.6d byte\n",
                                    outPipe, i, WriteLen, nBytesWrite);
    }

     if (pinBuf)
    {
        nBytesRead =0;

        /* read untill we get some answer from the cutter */
        /* the cutter will return nBytesRead = 0 as long as no data is
    available*/

        while (!nBytesRead)
        {
            success = ReadFile(hRead, pinBuf, ReadLen, &nBytesRead, NULL);
            /* add some kind of timeout or abort procedure ? */
        }

        // we have some data, process it
        pinBuf[nBytesRead] = 0;
```

```
        printf(pinBuf);
        printf("\n\r");
        delayms(20);      // add some delay to allow the cutter to prepare
  its next data

        TotalBytesRead = nBytesRead;

        /*if more data needed read, read more data,
         The cutter will return 0 bytes, while it is preparing its data !!
        */

        while (nBytesRead)
        {
              success = ReadFile(hRead, pinBuf, ReadLen, &nBytesRead, NULL);
              TotalBytesRead += nBytesRead;
              pinBuf[nBytesRead]=0;
              printf(pinBuf);
              delayms(20);
        }

        printf("\n<%s> R (%04.4d) : %06.6d bytes read\n", inPipe, i,
                                                          TotalBytesRead);
 }

 if (pinBuf)
 {
        free(pinBuf);
 }

 if (poutBuf)
 {
        free (poutBuf);
 }


 // close devices if needed
 if (hRead != INVALID_HANDLE_VALUE)
        CloseHandle(hRead);
 If (hWrite != INVALID_HANDLE_VALUE)
        CloseHandle(hWrite);


 // free the dll Library
 if (hinstLib)
        FreeLibrary(hinstLib);

 return 0;
}
```

7.6 Handshake Sample

The following application shows how to implement a handshaking method. It also uses the Run-Time Dynamic Linking method.
The program starts with creating a handle `fp` to the file specified in the command-line argument `argv[1]`. It then loads the library SummaUSB.dll and stores the handle in `hLib`. This handle is then used to retrieve the address of the function `open_file` to get a handle to an USB-pipe. This address `pOpenFile` is then used to get a handle to the USB output-pipe `hWrite` and status-pipe `hStatus`.
Now we are ready to transmit the data to the cutter.
First, we check if we're not at the end of a file, then we read a chunk of data (up to 256 bytes). If we could get the data, we have to check if there is space for it in the cutter's buffer. This is accomplished by reading form the USB status-pipe `hStatus` with the `ReadFile()` function. The data `szBuffer` returned represents the free space in the cutter's internal buffer. The converted value `nBufferFree` is compared against the amount of data `iCount` we've previously read. When there's not enough space for the data, we enter a loop until space becomes available. As soon as there's enough space for the data, we can send it to the cutter using the `WriteFile()` function with a handle to the USB output-pipe `hWrite`.
All this is repeated until all data has been sent to the cutter.
Next, all handles are closed and the Summa USB library `hLib` is unloaded.

```
// File2USB.cpp : Defines the entry point for the console application.

#include <windows.h>
#include <stdlib.h>
#include <stdio.h>

char     inPipe[32] = "PIPE00" ;  // pipe to read responces from the cutter
char    outPipe[32] = "PIPE01" ;  // pipe to send data to the cutter
char statusPipe[32] = "PIPE02" ;  // pipe to query free space in the cutter's
internal buffer

// prototype for open_file function from summausb.dll
typedef HANDLE (__stdcall * OPENFILE)(char *, DWORD) ;
OPENFILE pOpenFile ; // pointer to the open_file function

int main(int argc, char* argv[])
{
 FILE *fp ;
 HMODULE hLib ;              // The SummaUSB library handle
 HANDLE hWrite ;             // A handle to the USB output pipe
 HANDLE hStatus ;            // A handle to the USB status pipe
 char szData[256] ;          // The data buffer
 unsigned int iCount ;       // Amount of data we've read
 char szBuffer[16] ;
 unsigned long n ;
 unsigned long nBufferFree ; // Free space in the cutter's internal buffer
 BOOL bSuccess ;


 // A filename is required
 if (argc != 2)
 {
       printf("Usage : File2USB <filename>\n");
       return 1 ;
 }

 // Open the specified file
```

```
	fp = fopen(argv[1], "rb") ;

	if (fp == NULL)
	{
		printf("Error: could not open file \"%s\"\n", argv[1]) ;
		return 2 ;
	}

	// Load the library which connects our app to the USB driver
	hLib = LoadLibrary("SummaUSB.dll") ;

	if (hLib == NULL)
	{
		fclose(fp) ;
		printf("Error: could not open SummaUSB.dll\n") ;
		return 3 ;
	}

	// Get the address of the function which gives us a handle to the specified pipe
	pOpenFile = (OPENFILE)GetProcAddress(hLib, "open_file") ;

	if (pOpenFile == NULL)
	{
		fclose(fp) ;
		printf("Error: could not retrieve address of open_file function\n",
outPipe) ;
		return 4 ;
	}

	// Get a handle to the pipe where whe can put our data
	hWrite = (pOpenFile)(outPipe, 0) ;

	if (hWrite == INVALID_HANDLE_VALUE)
	{
		fclose(fp) ;
		printf("Error: could not open pipe \"%s\"\n", outPipe) ;
		return 5 ;
	}

	// Get a handle to the pipe where whe can read the free space in the cutter's
internal buffer
	hStatus = (pOpenFile)(statusPipe, 0) ;

	if (hStatus == INVALID_HANDLE_VALUE)
	{
		fclose(fp) ;
		printf("Error: could not open pipe \"%s\"\n", statusPipe) ;
		return 6 ;
	}

	bSuccess = TRUE ;

	// Repeat until no more data available or when there are problems on the USB port
	while (!feof(fp) && bSuccess)
	{
		// Get some data to send
		iCount = fread(szData, sizeof(char), sizeof(szData), fp) ;

		// If there is something to send
		if (iCount > 0)
		{
			// Use software handshaking
```

```
                do
                {
                        // Read data from USB status pipe
                        bSuccess = ReadFile(hStatus, szBuffer, sizeof(szBuffer), &n,
0) ;
                        szBuffer[n] = 0 ;               // Terminate string
                        nBufferFree = atol(szBuffer) ;  // Free space in internal buffer
of the cutter

                } while (bSuccess && nBufferFree < iCount) ; // Repeat until space
becomes available

                // Send the data through the USB output pipe
                bSuccess = WriteFile(hWrite, szData, iCount, &n, 0) ;
        }
 } ;

 CloseHandle(hStatus) ;
 CloseHandle(hWrite) ;
 fclose(fp) ;

 // Release the SummaUSB library
 FreeLibrary(hLib) ;
 return 0;
}
```

8 Appendix

8.1 Maximum Pressure and Speed by Model

The maximum tool-pressure, speed and media width depends on the cutter-model, and is summarized in the following table.

Device	Max. Pressure	Max. Speed	Max Cutting Width
S3D75	400 gr.	1000 mm/s	742 mm
S3D120	400 gr.	1000 mm/s	1200 mm
S3D140	400 gr.	1000 mm/s	1350 mm
S3D160	400 gr.	1000 mm/s	1580 mm
S3T75	1000 gr.	1000 mm/s	742 mm
S3T120	1000 gr.	1000 mm/s	1200 mm
S3T140	1000 gr.	1000 mm/s	1350 mm
S3T160	1000 gr.	1000 mm/s	1580 mm
S3TC75	1000 gr.	1000 mm/s	742 mm
S3TC160	1000 gr.	1000 mm/s	1580 mm

Table 8.1 - Maximum Pressure And Speed By Model