

## **SilverPakT Integrated Controller/Driver and Motor**

**Commands Guide**

Version 1.00

---

RMS Technologies  
2533 N. Carson St. #4698, Carson City, NV 89706-0147  
1-877-301-3609

[www.rmsmotion.com](http://www.rmsmotion.com)

[sales@rmsmotion.com](mailto:sales@rmsmotion.com)

Thank you for purchasing the SilverPakT. This product is warranted to be free of manufacturing defects for one year from the date of purchase.

**PLEASE READ BEFORE USING**

Before you start, you must have a suitable step motor, a DC power supply suitable for the motor. The power supply voltage must be between 4 times and 20 times the motor's rated voltage.

**DISCLAIMER**

The information provided in this document is believed to be reliable. However, no responsibility is assumed for any possible inaccuracies or omissions. Specifications are subject to change without notice.

RMS Technologies reserves the right to make changes without further notice to any products herein to improve reliability, function, or design. RMS Technologies does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights, nor the rights of others.

## Table of Contents

<b>DISCLAIMER</b> .....	<b>2</b>
<b>1. SILVERPAK T SETUP</b> .....	<b>5</b>
List of Parts: .....	5
<b>2. PROGRAMMING THE SILVERPAK T</b> .....	<b>5</b>
<b>3. COMMAND TABLES</b> .....	<b>8</b>
MOTION COMMANDS.....	8
PARAMETER COMMANDS.....	8
I/O Port COMMANDS .....	8
CONTROL COMMANDS.....	8
CALCULATION COMMANDS.....	8
<b>4. SYNTAX</b> .....	<b>9</b>
<b>5. BASIC TMCL CONCEPTS</b> .....	<b>9</b>
5.1 Binary command format .....	9
5.2 Checksum calculation.....	10
5.3 The reply format .....	10
5.4 Status code .....	11
5.5 Stand-alone applications .....	11
<b>6. COMMANDS</b> .....	<b>12</b>
<b>ROR – Rotate Right</b> .....	<b>13</b>
<b>ROL – Rotate Left</b> .....	<b>14</b>
<b>MST – Motor Stop</b> .....	<b>15</b>
<b>MVP – Move to Position</b> .....	<b>16</b>
<b>SAP – Set Axis Parameters</b> .....	<b>17</b>
<b>GAP – Get Axis Parameters</b> .....	<b>18</b>
<b>STAP – Store Axis Parameter</b> .....	<b>19</b>
<b>RSAP – Restore Axis Parameter</b> .....	<b>20</b>
<b>SGP – Set Global Parameter</b> .....	<b>21</b>
<b>GGP – Get Global Parameter</b> .....	<b>22</b>
<b>STGP – Store Global Parameter</b> .....	<b>23</b>
<b>RSGP – Restore Global Parameter</b> .....	<b>24</b>
<b>RFS – Reference Switch</b> .....	<b>25</b>
<b>SIO – Set Input/Output</b> .....	<b>26</b>
<b>GIO – Get Input/Output</b> .....	<b>27</b>
<b>CALC – Calculate</b> .....	<b>28</b>
<b>COMP - Compare</b> .....	<b>29</b>
<b>JC – Jump Conditional</b> .....	<b>30</b>
<b>JA – Jump Always</b> .....	<b>31</b>
<b>CSUB – Call Subroutine</b> .....	<b>32</b>
<b>WAIT – Wait for an event to occur</b> .....	<b>33</b>
<b>STOP – Stop the TMCL Program</b> .....	<b>34</b>
<b>SCO – Set Coordinate</b> .....	<b>35</b>
<b>GCO – Get Coordinate</b> .....	<b>36</b>
<b>CCO – Capture Coordinate</b> .....	<b>37</b>
<b>CALCX – Calculate using the X register</b> .....	<b>38</b>

<b>AAP – Accumulator to Axis Parameter</b> .....	<b>39</b>
<b>AGP – Accumulator to Global Parameter</b> .....	<b>40</b>
<b>CLE – Clear Error Flags</b> .....	<b>41</b>
<b>User Definable Commands (UF0 to UF7)</b> .....	<b>42</b>
<b>TMCL Control Functions</b> .....	<b>43</b>
<b>Axis Parameters</b> .....	<b>44</b>
Basic Parameters: .....	44
Advanced Parameters: .....	44
<b>Global Parameters</b> .....	<b>46</b>
Bank 0 .....	46
Bank 1 .....	47
Bank 2 .....	48
<b>Reference Searching</b> .....	<b>48</b>
<b>Stall Detection</b> .....	<b>49</b>
<b>Fixing microstep errors</b> .....	<b>49</b>
<b>The syntax of TMCL in the TMCL assembler</b> .....	<b>49</b>
Assembler directives .....	49
Symbolic constants .....	49
Constant expressions .....	50
Labels .....	50
Comments .....	51
TMCL Commands .....	51
<b>Sample code</b> .....	<b>51</b>

## 1. SILVERPAK T SETUP

### List of Parts:

- SilverPakT unit
- DB15 Cable
- USB485 converter card (optional)
- Power supply +7 to 28VDC
- PC

Follow the schematic for correct connection:

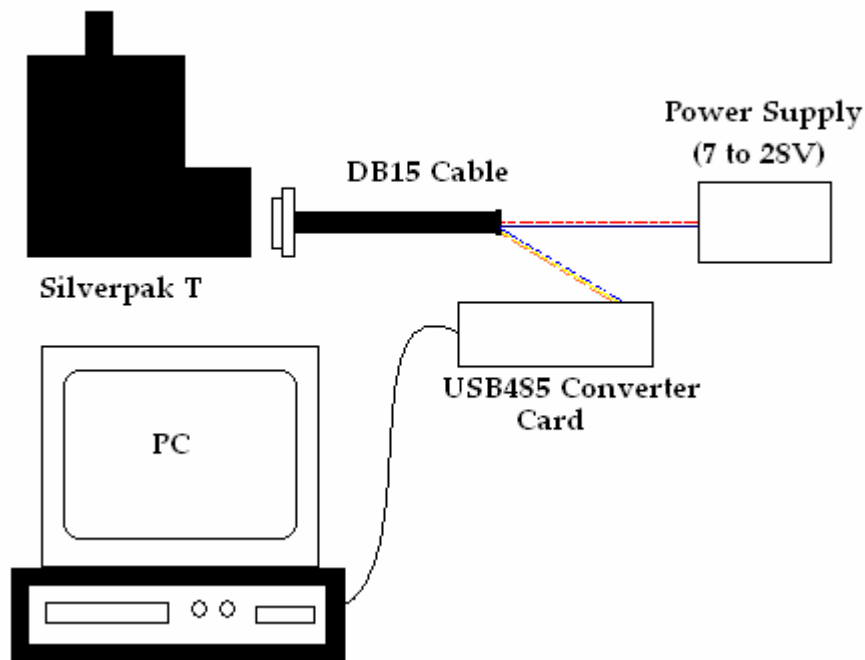


Figure 3: Connection Diagram

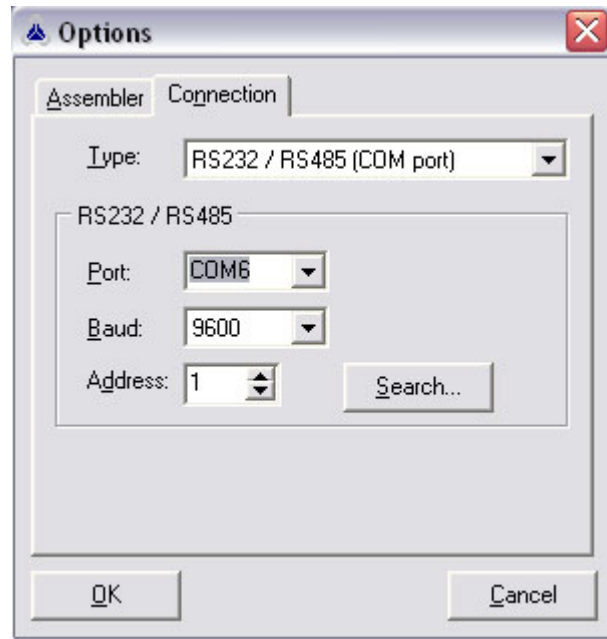
## 2. PROGRAMMING THE SILVERPAK T

1. Download the TMCL Program. Example programs are included. One of the sample programs is labeled as 'Motor Config.tmc'. Open this program using the TMCL program

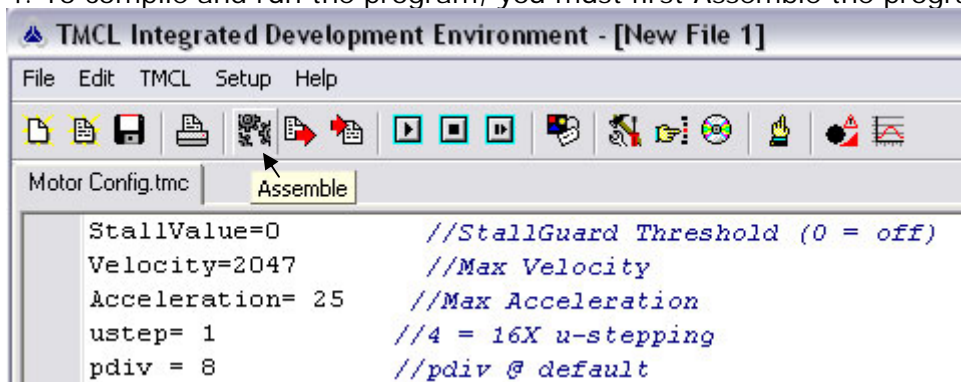
2. 'Motor Config.tmc' will show basic setup for microstepping, StallGuard Threshold, velocities, output current, acceleration, and decay modes. This program will rotate right for 1000 steps. Please be sure the motor shaft is not attached to a device that may cause harm.

3. To setup the correct COM port, go to Setup → Options → Connection. Select the correct COM port.

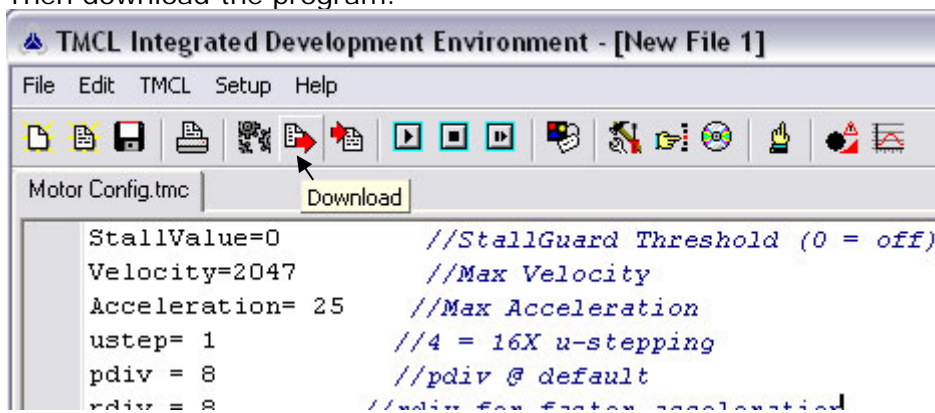
To ensure you have the right COM port, right click on My Computer. Go to Properties. Then click on the Hardware tab. Click on Device Manager. You should see a list of COM settings. Please verify with the list.



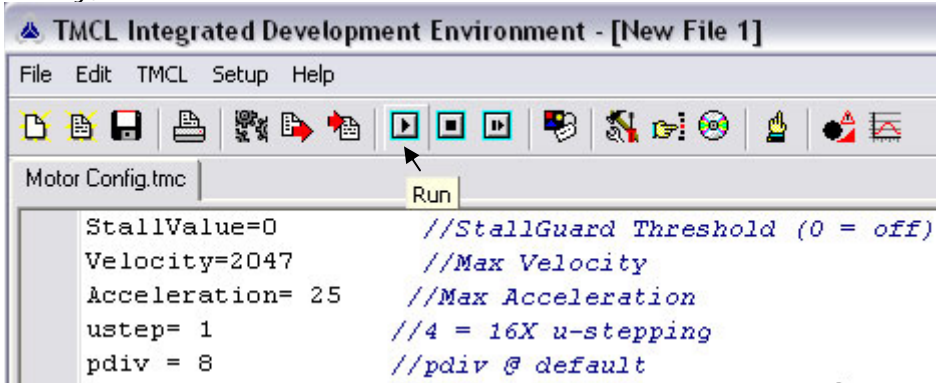
4. To compile and run the program, you must first Assemble the program:



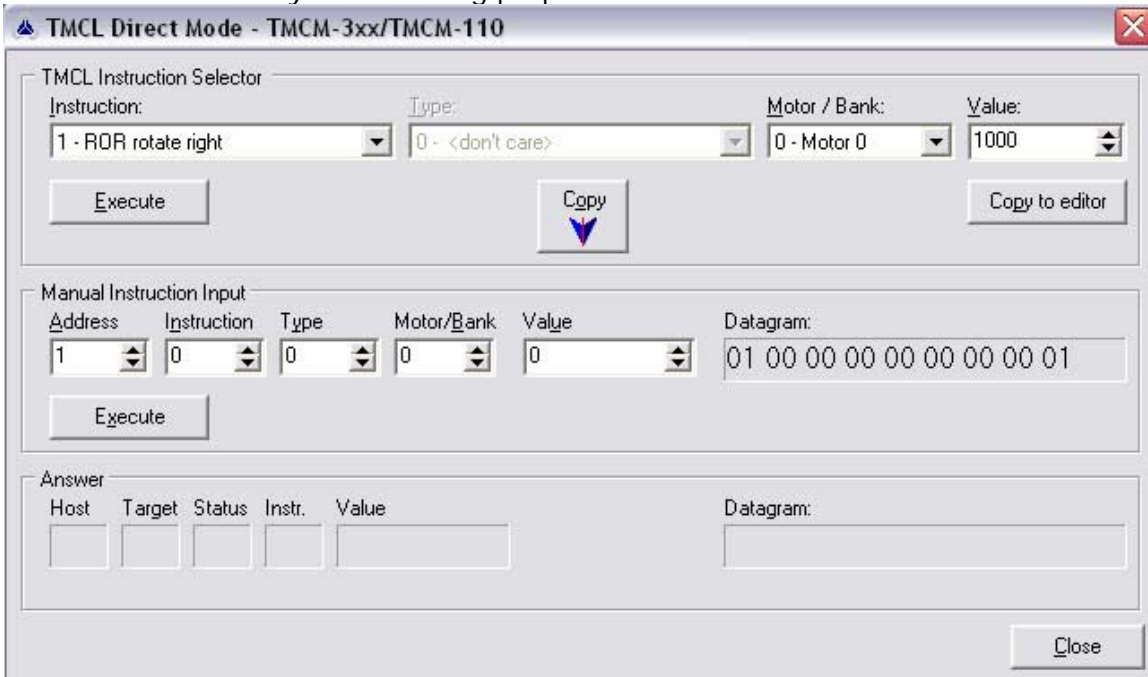
Then download the program:



Finally, click on the Run button:



5. To run the SilverpakT in Direct Mode, go to TMCL → Direct Mode. Commands can be executed on the fly to for testing purposes.



### 3. COMMAND TABLES

The following tables break down the Trinamic Motion Control Language (TMCL) with corresponding commands for ease of use.

#### MOTION COMMANDS

Mnemonic	Meaning
ROL	Rotate Left
ROR	Rotate Right
MVP	Move To Position
MST	Motor Stop
RFS	Reference Switch
SCO	Store Coordinate
CCO	Capture Coordinate
GCO	Get Coordinate

#### PARAMETER COMMANDS

Axis parameters can be set independently for every axis, whereas global parameters control the behavior of the module itself.

Mnemonic	Meaning
SAP	Set Axis Parameter
GAP	Get Axis Parameter
STAP	Store Axis Parameter in EEPROM
RSAP	Restore Axis Parameter in EEPROM
SGP	Set Global Parameter
GGP	Get Global Parameter
STGP	Store Global Parameter in EEPROM
RSGP	Restore Global Parameter in EEPROM

#### I/O Port COMMANDS

Mnemonic	Meaning
SIO	Set Output
GIO	Get Input
SAC	Access to External SPI Device

#### CONTROL COMMANDS

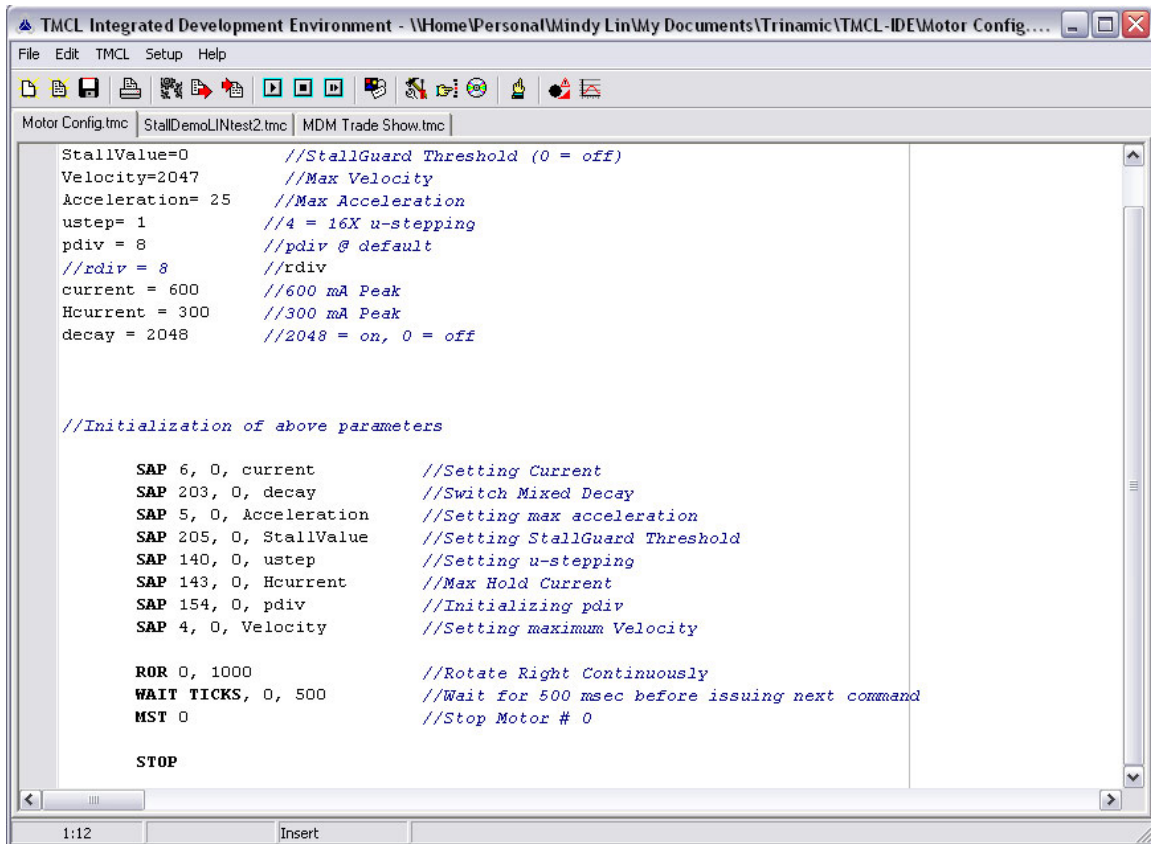
Mnemonic	Meaning
JA	Jump Always
JC	Jump Conditional
COMP	Compare accumulator with constant value
CLE	Clear Error Flags
CSUB	Call Subroutine
RSUB	Return from Subroutine
WAIT	Wait for a specified event
STOP	End of a TMCL Program

#### CALCULATION COMMANDS

Mnemonic	Meaning
CALC	Calculate using the accumulator and a constant value
CALCX	Calculate using the accumulator and the X register
AAP	Copy accumulator to an axis parameter
AGP	Copy accumulator to a global parameter

## 4. SYNTAX

The SilverpakT uses the Trinamic TMCL (Trinamic Motion Control Language). Syntax is easy to follow.



```
StallValue=0 //StallGuard Threshold (0 = off)
Velocity=2047 //Max Velocity
Acceleration= 25 //Max Acceleration
ustep= 1 //4 = 16X u-stepping
pdiv = 8 //pdiv @ default
//rdiv = 8 //rdiv
current = 600 //600 mA Peak
Hcurrent = 300 //300 mA Peak
decay = 2048 //2048 = on, 0 = off

//Initialization of above parameters

SAP 6, 0, current //Setting Current
SAP 203, 0, decay //Switch Mixed Decay
SAP 5, 0, Acceleration //Setting max acceleration
SAP 205, 0, StallValue //Setting StallGuard Threshold
SAP 140, 0, ustep //Setting u-stepping
SAP 143, 0, Hcurrent //Max Hold Current
SAP 154, 0, pdiv //Initializing pdiv
SAP 4, 0, Velocity //Setting maximum Velocity

ROR 0, 1000 //Rotate Right Continuously
WAIT TICKS, 0, 500 //Wait for 500 msec before issuing next command
MST 0 //Stop Motor # 0

STOP
```

- Variables can be named and reused throughout the program
- Most parameters have acronyms that are intuitive, for example, SAP is 'Set Axis Parameters'.
- The syntax follows a three letter acronym, followed by two or three parameters. Typically, the parameters will be:

Instruction	Type	Motor No / Bank	Value
SAP	5	0	100

This example will set the acceleration to value 100 for Motor number 0.

## 5. BASIC TMCL CONCEPTS

### 5.1 Binary command format

Every command has a mnemonic and a binary representation. When commands are sent from a host to a module, the binary format has to be used. Every command consists of a one-byte command field, a one-byte type field, a one-byte motor/bank field and a four-byte value field. So the binary representation of a command always has seven bytes. When a command is to be sent via RS485 interface, it has to be enclosed by an address byte at the beginning and a checksum byte at the end. So it then consists of nine bytes. The checksum is calculated by

adding up all the other bytes using an 8-bit addition. So the binary command format when using RS485 is as follows:

Bytes	Meaning
1	Module address
1	Command number
1	Type number
1	Motor or Bank number
4	Value (MSB first)
1	Checksum

## 5.2 Checksum calculation

As mentioned above, the checksum is calculated by adding up all bytes (including the module address byte) using 8-bit addition. Here are two examples to show how to do this:

- in C:

```
unsigned char i, Checksum;
unsigned char Command[9];
//Set the "Command" array to the desired command
Checksum = Command[0];
for(i=1; i<8; i++)
Checksum+=Command[i];
Command[8]=Checksum; //insert checksum as last byte of the command
//Now, send it to the module
```

- in Delphi:

```
var
i, Checksum: byte;
Command: array[0..8] of byte;
//Set the "Command" array to the desired command
//Calculate the Checksum:
Checksum:=Command[0];
for i:=1 to 7 do Checksum:=Checksum+Command[i];
Command[8]:=Checksum;
//Now, send the "Command" array (9 bytes) to the module
```

## 5.3 The reply format

Every time a command has been sent to a module, the module sends a reply:

Bytes	Meaning
1	Reply Address
1	Module Address
1	Status (e.g. 100 means "no error")
1	Command Number
4	Value (MSB first)
1	Checksum

The checksum is also calculated by adding up all the other bytes using an 8-bit addition. **Do not send the next command before you have received the reply.**

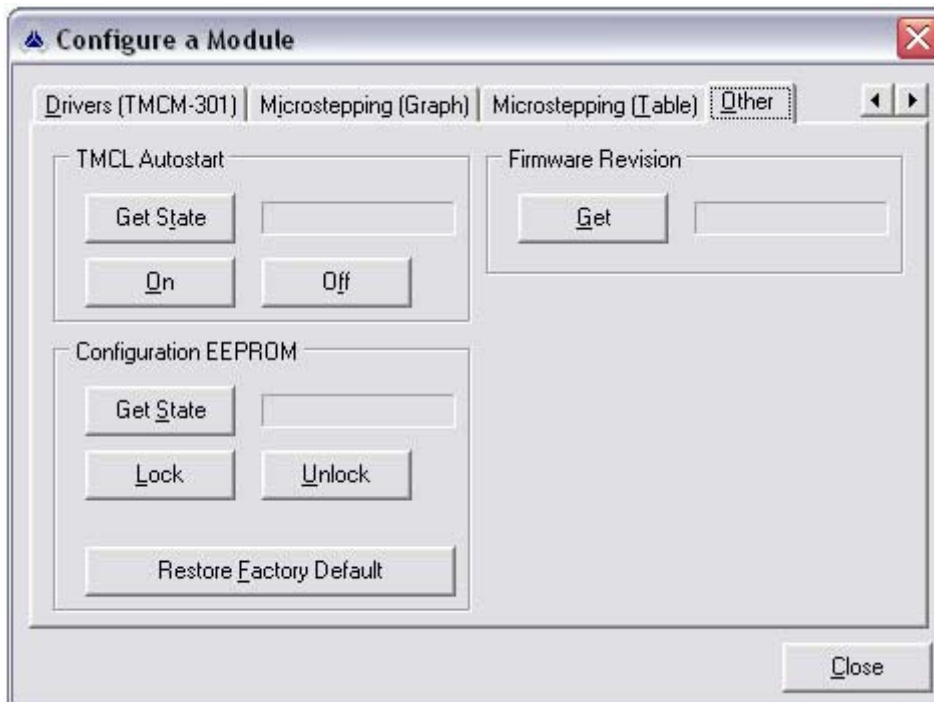
## 5.4 Status code

The reply contains a status code. This status code can have one of the following values:

Code	Meaning
100	Successfully executed, no error
101	Command loaded into TMCL program EEPROM
1	Wrong checksum
2	Invalid command
3	Wrong type
4	Invalid value
5	Configuration EEPROM locked
6	Command not available

## 5.5 Stand-alone applications

The TMCL program can be used to develop stand-alone TMCL applications that can be downloaded into the EEPROM and can be run without the use of a PC. The program contains an editor and a "TMCL assembler" where the commands can be entered using the correct syntax and format. The code can then be downloaded into the module to be executed there. To run the program upon power up, go to Setup → Configure Module → Other. Then choose "ON" for TMCL Autostart:



## 6. COMMANDS

Commands are listed by their command name. Each page explains in detail the description of the command, what occurs internally, related commands, syntax, and an example. In addition, binary representation and control replies are given for customers using their own PLC to communicate to the SilverpakT. Finally, notes are written for particular commands if applicable.

There is one command per page, beginning with the following page.

## ROR – Rotate Right

**Description:** This instruction begins rotation in the “right” direction, i.e. increasing the position counter.

**Internal function:** First, velocity mode is selected. Then, the velocity value is transferred to axis parameter ‘target velocity’.

**Related commands:** ROL, MST, SAP, GAP

**Syntax:** ROR <motor number>, <velocity>

**Example:** Rotate right at a velocity of 1000

ROR 0, 1000

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
1	(don't care)	0	0 to 8191

### Reply in direct mode:

Status	Value
100 – OK	(don't care)

### Binary: ROR 0, 350

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$01	\$00	\$00	\$00	\$00	\$01	\$5e	\$61

### Notes:

If the speed is creating resonance or is not as expected, adjust the pulse divisor (pdiv) value. Also try different values for decay mode (damping). Depending on motor characteristics, a decay value of 1000 dampens resonance best.

Turn on 100% decay:                   SAP 203, 0, 2048  
 Turn off decay:                         SAP 203, 0, 0

## ROL – Rotate Left

**Description:** This instruction begins rotation in the “left” direction, i.e. decreasing the position counter.

**Internal function:** First, velocity mode is selected. Then, the velocity value is transferred to axis parameter ‘target velocity’.

**Related commands:** ROR, MST, SAP, GAP

**Syntax:** ROL <motor number>, <velocity>

**Example:** Rotate left at a velocity of 1200

ROL 0, 1200

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
2	(don't care)	0	0 to 8191

### Reply in direct mode:

Status	Value
100 – OK	(don't care)

### Binary: ROL 0, 1200

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$02	\$00	\$00	\$00	\$00	\$04	\$b0	\$b7

### Notes:

If the speed is creating resonance or is not as expected, adjust the pulse divisor (pdiv) value. Also try different values for decay mode (damping). Depending on motor characteristics, a decay value of 1000 dampens resonance best.

Turn on 100% decay:               SAP 203, 0, 2048  
 Turn off decay:                    SAP 203, 0, 0

# MST – Motor Stop

**Description:** This instruction stops the motor from rotation.

**Internal function:** The axis parameter 'target velocity' is set to zero.

**Related commands:** ROR, ROL, SAP, GAP

**Syntax:** MST <motor number>

**Example:** Stop Motor

MST 0

**Binary Representation:**

Instruction No.	Type	Motor / Bank	Value
3	(don't care)	0	(don't care)

**Reply in direct mode:**

Status	Value
100 – OK	(don't care)

**Binary: MST 0**

Byte Index	0	1	2	3	4	5	6	7	8
<b>Function</b>	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
<b>Value (hex)</b>	\$01	\$03	\$00	\$00	\$00	\$00	\$00	\$00	\$04

**Notes:**

MST can be used to have a motor run for a specified time, then issue a MST command:

```
ROR 0, 1000 //Rotate right with a velocity of 1000
WAIT TICKS, 0, 500 //Wait for 500 milliseconds
MST 0 //Stop Motor
```

MST can also be used to rotate the motor and wait upon a signal to stop the motor:

```

Loop:   ROR 0, 1000 //Rotate Right Continuously
        CSUB CheckIO //Call subroutine "CheckIO"
        JA Loop //Loop continuously, check for output continuously
        STOP //Stop Program

CheckIO:
        GAP 11, 0 //Get axis parameter 'left limit switch status'
        COMP 1 //Compare if left limit switch is a '1'
        JC EQ, StopMotor //Jump to 'StopMotor' if previous statements are equal
        RSUB //If no jump, return back to original program above

StopMotor:
        MST 0 //Stop Motor
        RSUB //Return to original program above

```



## SAP – Set Axis Parameters

**Description:** This sets all the axis parameters needed to run the motor properly. Chapter 6 describes each axis parameter setting in detail. These apply to SAP, GAP, STAP, RSAP, and AAP commands.

**Internal function:** The parameter format is converted ignoring leading zeros and negative values. The parameter is then transferred to the correct position.

**Related commands:** GAP, STAP, RSAP, AAP

**Syntax:** SAP <parameter number>, <motor number>, <value>

**Example:** Set the current to 200 mA

SAP 6, 0, 200

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
5	<parameter no>	0	<value>

### Reply in direct mode:

Status	Value
100 – OK	(don't care)

### Binary: SAP 6, 0, 200

Byte Index	0	1	2	3	4	5	6	7	8
<b>Function</b>	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
<b>Value (hex)</b>	\$01	\$05	\$06	\$00	\$00	\$00	\$00	\$c8	\$d4

### Notes:

The most common axis parameters you will use are listed below, with example values:

```

SAP 6, 0, 500           //Setting Current to 0.5 Amps/Phase
SAP 203, 0, 1000       //Switch to Mixed Decay value of 1000
SAP 5, 0, 1100        //Setting max acceleration to 1100
SAP 205, 0, 3         //Setting StallGuard Threshold to medium
SAP 140, 0, 4         //Setting u-stepping to 16x
SAP 143, 0, 250       //Max Hold Current to 0.25 Amps/Phase
SAP 154, 0, 4         //For a 1.8° Stepper, pdiv=4 and vel=1049 is 5 RPS
SAP 4, 0, 1094        //For a 1.8° Stepper, pdiv=4 and vel=1049 is 5 RPS
SAP 138, 0, 11       //Ramp Divisor, the higher the number, the faster acceleration
STOP
    
```

## GAP – Get Axis Parameters

**Description:** This gets all the axis parameters. Chapter 6 describes each axis parameter setting in detail. These apply to SAP, GAP, STAP, RSAP, and AAP commands.

**Internal function:** The parameter is read out of the correct position in the appropriate device. The parameter format is converted adding leading zeros (or ones for negative values).

**Related commands:** SAP, STAP, RSAP, AAP

**Syntax:** GAP <parameter number>, <motor number>

**Example:** Get the actual position of motor

GAP 0, 1

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
6	<parameter no>	0	(don't care)

### Reply in direct mode:

Status	Value
100 – OK	(don't care)

### Binary: GAP 0, 1

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$06	\$01	\$00	\$00	\$00	\$00	\$00	\$08

### Reply: status=no error, position=711

Byte Index	0	1	2	3	4	5	6	7	8
Function	Host – address	Target – address	Status	Instruction	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$02	\$01	\$64	\$06	\$00	\$00	\$02	\$c7	\$36

## STAP – Store Axis Parameter

**Description:** Axis parameters are located in RAM memory, so modifications are lost at power down. This instruction enables permanent storing. Most parameters are automatically restored after power up. Chapter 6 describes each axis parameter setting in detail. These apply to SAP, GAP, STAP, RSAP, and AAP commands.

**Internal function:** The specified parameter is copied from its RAM location the configuration EEPROM.

**Related commands:** GAP, SAP, RSAP, AAP

**Syntax:** STAP <parameter number>, <motor number>

**Example:** Store the maximum speed of the motor into EEPROM.

STAP 4, 0

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
7	<parameter no>	0	(don't care)

### Reply in direct mode:

Status	Value
100 – OK	(don't care)

### Binary: STAP 4, 0

Byte Index	0	1	2	3	4	5	6	7	8
<b>Function</b>	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
<b>Value (hex)</b>	\$01	\$07	\$04	\$00	\$00	\$00	\$00	\$00	\$0c

**Note:** The STAP command will not have any effect when the configuration EEPROM is locked

## RSAP – Restore Axis Parameter

**Description:** For all configuration-related axis parameters, non-volatile memory locations are provided. By default, most parameters are automatically restored after power up. A single parameter that has been changed before can be reset by this instruction. Chapter 6 describes each axis parameter setting in detail. These apply to SAP, GAP, STAP, RSAP, and AAP commands.

**Internal function:** The specified parameter is copied from the configuration EEPROM memory to its RAM location.

**Related commands:** GAP, SAP, STAP, AAP

**Syntax:** RSAP <parameter number>, <motor number>

**Example:** Restore the maximum current of the motor

RSAP 6, 0

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
8	<parameter no>	0	(don't care)

### Reply in direct mode:

Status	Value
100 – OK	(don't care)

### Binary: RSAP 6, 0

Byte Index	0	1	2	3	4	5	6	7	8
<b>Function</b>	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
<b>Value (hex)</b>	\$01	\$08	\$06	\$00	\$00	\$00	\$00	\$00	\$0f

## SGP – Set Global Parameter

**Description:** Global parameters are related to the host interface, peripherals or application specific variables. The different groups of these parameters are organized in "banks" to allow a larger total number for future products. Currently, only bank 0 and 1 are used for global parameters, and bank 2 is used for user variables. See Chapter 7 for a complete list of parameters.

**Internal function:** The parameter format is converted ignoring leading zeros (or ones for negative values). The parameter is transferred to the correct position.

**Related commands:** GGP, STGP, RSGP, AGP

**Syntax:** SGP <parameter number>, <bank number>, <value>

**Example:** Set the Baud Rate to 57600

SGP 65, 0, 5

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
9	<parameter no>	0	<value>

### Reply in direct mode:

Status	Value
100 – OK	(don't care)

### Binary: SGP 65, 0, 5

Byte Index	0	1	2	3	4	5	6	7	8
<b>Function</b>	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
<b>Value (hex)</b>	\$01	\$09	\$41	\$00	\$00	\$00	\$00	\$05	\$50

## GGP – Get Global Parameter

**Description:** All global parameters can be read with this function. This is copied to the accumulator register for further processing purposes such as conditional jumps. In direct mode, the result is also output in the “value” field of the reply. See Chapter 7 for a complete list of parameters.

**Internal function:** The parameter is read out of the correct position in the appropriate device. The parameter format is converted adding leading zeros (or ones for negative values).

**Related commands:** SGP, STGP, RSGP, AGP

**Syntax:** GGP <parameter number>, <bank number>

**Example:** Get the Baud Rate

GGP 65, 0

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
10	<parameter no>	0	(don't care)

### Reply in direct mode:

Status	Value
100 – OK	(don't care)

### Binary: GGP 65, 0

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$0a	\$41	\$00	\$00	\$00	\$00	\$00	\$4c

### Reply: status=no error, Value = 1

Byte Index	0	1	2	3	4	5	6	7	8
Function	Host – address	Target – address	Status	Instruction	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$02	\$01	\$64	\$0a	\$00	\$00	\$02	\$01	\$74

## STGP – Store Global Parameter

**Description:** Some global parameters are located in RAM memory, so modifications are lost at power down. This instruction enables permanent storing. Most parameters are automatically restored after power up. See Chapter 7 for a complete list of parameters.

**Internal function:** The specified parameter is copied from its RAM location to the configuration EEPROM.

**Related commands:** SGP, GGP, RSGP, AGP

**Syntax:** SGP <parameter number>, <bank number>

**Example:** Store the baud rate of the motor

STGP 65, 0

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
11	<parameter no>	0	<value>

### Reply in direct mode:

Status	Value
100 – OK	(don't care)

### Binary: SGP 65, 0, 5

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$0b	\$41	\$00	\$00	\$00	\$00	\$00	\$4d

## RSGP – Restore Global Parameter

**Description:** This instruction recovers the (permanently) stores the value of a RAM-located parameter. See Chapter 7 for a complete list of parameters.

**Internal function:** The specified parameter is copied from the configuration EEPROM to its RAM location.

**Related commands:** SGP, GGP, STGP, AGP

**Syntax:** RSGP <parameter number>, <bank number>

**Example:** Restore the baud rate

RSGP 65, 0

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
12	<parameter no>	0	(don't care)

### Reply in direct mode:

Status	Value
100 – OK	(don't care)

### Binary: GGP 65, 0

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$0c	\$41	\$00	\$00	\$00	\$00	\$00	\$4e

## RFS – Reference Switch

**Description:** A build-in reference point search algorithm can be started (and stopped). The reference search algorithm provides switching point calibration and three switch modes. The status of the reference search can also be queried to see if it has already finished. (In a TMCL program it is better to use the WAIT command to wait for the end of a reference search.) Please see the appropriate parameters in the axis parameter table to configure the reference search algorithm to meet your needs (Chapter 6). The reference search can be started or stopped, or the actual status of the reference search can be checked.

**Internal function:** The reference search is implemented as a state machine, so interaction is possible during execution.

**Related commands:** WAIT

**Syntax:** RFS <START|STOP|STATUS>, <motor number>

**Example:** Start Reference search for motor 0

RFS START, 0

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
13	START (0)	0	(don't care)
	STOP (1)	0	(don't care)
	STATUS (2)	0	(don't care)

### Reply in direct mode:

When using type 0 (START) or 1 (STOP):

Status	Value
100 – OK	(don't care)

When using type 2 (STATUS):

Status	Value
100 – OK	0 – no reference search active. Other value – reference search is active.

### Binary: RFS START, 0

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$0d	\$00	\$00	\$00	\$00	\$00	\$00	\$0e

## SIO – Set Input/Output

**Description:** This command sets the status of a digital output either to low (0) or to high (1). Please see the outputs listed below.

**Internal function:** The passed value is transferred to the specified output line.

**Related commands:** GIO, WAIT

**Syntax:** SIO <port number>, <bank number>, <value>

**Example:** Turn on Output 1 (on motor # 0)

SIO 0, 0, 1

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
14	<port number>	<bank number>	0 or 1

### Reply in direct mode:

Status	Value
100 – OK	(don't care)

### Binary: SIO 0, 0, 1

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$0e	\$00	\$00	\$00	\$00	\$00	\$01	\$10

### Port:

Type	I/O line	Range
0	DOUT0	0/1

## GIO – Get Input/Output

**Description:** This function reads a digital or analogue input port. So, digital lines will read 0 and 1, while the ADC channels deliver their 10 bit result in the range of 0...1023. The requested value is copied to the "accumulator" (accu) for further processing purposes such as conditioned jumps. In direct mode the value is also output in the "value" field of the reply. The actual status of a digital output line can also be read.

**Internal function:** The specified line is read and the value is returned in the reply and copied to the accumulator.

**Related commands:** SIO, WAIT

**Syntax:** GIO <port number>, <bank number>

**Example:** Get the analog value of ADC Channel 1 (on motor # 0)

GIO 1, 1

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
15	<port number>	<bank number>	(don't care)

### Reply in direct mode:

Status	Value
100 – OK	<status of the port>

### Binary: GIO 1, 1

Byte Index	0	1	2	3	4	5	6	7	8
<b>Function</b>	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
<b>Value (hex)</b>	\$01	\$0f	\$01	\$01	\$00	\$00	\$00	\$00	\$12

### Reply

Byte Index	0	1	2	3	4	5	6	7	8
<b>Function</b>	Host – address	Target – Address	Status	Instruction	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
<b>Value (hex)</b>	\$02	\$01	\$64	\$0f	\$00	\$00	\$01	\$fa	\$72

Value → 506

### Overview of available input ports:

I/O bank 0 – digital inputs. The ADIN lines can be read as digital or analog inputs at the same time. The analog values can be accessed in bank 1.

Type	I/O Line	Value
0	ADIN 0	0 or 1

I/O bank 1 – analog inputs. The ADIN lines can be read as digital or analog inputs at the same time. The digital states can be accessed in bank 0.

Type	I/O Line	Value
0	ADIN 0	0 to 1023

I/O bank 2 – here the status of the digital outputs can be read back

## CALC – Calculate

**Description:** A value in the accumulator variable, previously read by a function such as GAP (get axis parameter), can be modified with this instruction. Nine different arithmetic functions can be chosen and one constant operand value must (in most cases) be specified. The result is written back to the accumulator, for further processing like comparisons or data transfer.

**Related commands:** CALCX, COMP, JC, AAP, AGP, GAP, GGP, GIO

**Syntax:** CALC <op>, <value> (where op is ADD, SUB, MUL, DIV, MOD, AND, OR, XOR, NOT, or LOAD)

**Example:** Multiply accu by -5000

CALC MUL,-5000

### Binary Representation:

Instruction No.	Type	Value
19	0 ADD – add to accu 1 SUB – subtract from accu 2 MUL – multiply accu by 3 DIV – divide accu by 4 MOD – modulo divide by 5 AND – logical and accu with 6 OR – logical or accu with 7 XOR – logical exor accu with 8 NOT – logical invert accu 9 LOAD – load operand to accu	<operand>

### Binary: CALC MUL, -5000

Byte Index	0	1	2	3	4	5	6	7	8
<b>Function</b>	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
<b>Value (hex)</b>	\$01	\$13	\$02	\$00	\$FF	\$FF	\$EC	\$78	\$78

## COMP - Compare

**Description:** The specified number is compared to the value in the accumulator register. The result of the comparison can for example be used by the conditional jump (JC) instruction. This command is intend for use in stand-alone operation only and must not be used indirect mode.

**Internal function:** The specified value is compared to the internal "accumulator", which holds the value of a preceding "get" or calculate instruction (see GAP/GGP/GIO/CALC/CALCX). The internal arithmetic status flags are set according to the comparison result.

**Related commands:** JC (jump conditional), GAP, GGP, GIO, CALC, CALCX

**Syntax:** COMP <comparison value>

**Example:** Jump to the address given by the label when the motor position is greater than or equal to 1000. When it jumps to the label, it will turn on the LED connected to the output.

```
GAP 1, 0, 0    //get axis parameter type 1 (actual position), for motor #0, value 0 (don't care)
COMP 1000     //compare actual position to 1000
JC GE, Label1 //Jump if greater than or equal to (GE), go to address of label: Label1
STOP         //Stop TMCL program (after it has returned from the subroutine below
```

```
Label1: SIO 0, 0, 1 //Set I/O on motor to be high (1)
        RSUB       //Return from subroutine – return to original program above
```

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
20	(don't care)	(don't care)	<comparison value>

### Binary: COMP 1000

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$14	\$00	\$00	\$00	\$00	\$03	\$e8	\$00

## JC – Jump Conditional

**Description:** A conditional jump to a fixed address in the TMCL-program memory, if the specified condition is met. The conditions refer to the result of a preceding comparison, e.g. by the COMP instruction. For stand-alone operation only.

**Internal function:** The TMCL program counter is set to the passed value if the arithmetic status flags are in the appropriate state(s).

**Related commands:** JA, COMP, WAIT, CLE

**Syntax:** JC <condition>, <label>

**Example:** Jump to address when condition in previous line is met

```
GAP 1, 0, 0 //get axis parameter type 1 (actual position), for motor #0, value 0 (don't care)
COMP 1000 //compare actual position to 1000
JC GE, Label1 //Jump if greater than or equal to (GE), go to address of label: Label1
STOP //Stop TMCL program (after it has returned from the subroutine below)
```

```
Label1: ROL, 0, 1000 //Rotate left continuously at a speed of 1000
        RSUB //Return from subroutine – return to original program above
```

### Binary Representation:

Instruction No.	Type	Motor/Bank	Value
21	0 ZE - zero 1 NZ – not zero 2 EQ – equal 3 NE – not equal 4 GT – greater than 5 GE – greater than or equal 6 LT – less than 7 LE – less than or equal 8 ETO – time out error 9 EAL – external alarm 10 EDV – deviation error 11 EPO – position error 12 ESD – shutdown error	(don't care)	<jump address>

### Binary: JC GE, Label1 (assuming the label 'Label1' is at address 10)

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$15	\$05	\$00	\$00	\$00	\$00	\$0a	\$25

## JA – Jump Always

**Description:** Jump to a fixed address in the TMCL program memory. This command is intended for standalone operation only and not to be used in direct mode.

**Internal function:** The TMCL program counter is set to the passed value.

**Related commands:** JC, WAIT, CSUB

**Syntax:** JA <Label>

**Example:** An infinite loop in TMCL

```
Loop: MVP ABS, 0, 10000 //Move to absolute position 10000
      WAIT POS, 0, 0 //Wait until position is reached
      MVP ABS, 0, 0 //Move to absolute position 0
      WAIT POS, 0, 0 //Wait until position is reached
      JA Loop //Jump to the label "Loop"
```

**Binary: JA Loop** (assuming the label 'Loop' is at address 20)

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$16	\$00	\$00	\$00	\$00	\$00	\$14	\$2b

## CSUB – Call Subroutine

**Description:** Calls a subroutine in the TMCL program memory. This command is intended for standalone operation only and must not be used in direct mode.

**Internal function:** The actual TMCL program counter value is saved to an internal stack, then overwritten with the passed value. The number of entries in the internal stack is limited to 8. This also limits nesting of subroutine calls to 8. The command will be ignored if there is no more stack space left.

**Related commands:** RSUB, JA

**Syntax:** CSUB <Label>

**Example:** Call a subroutine

```

Loop: MVP ABS, 0, 10000 //Move motor to absolute position 10000
      CSUB SubW //Save program counter and jump to label "SubW"
      MVP ABS, 0, 0 //After returning from subroutine, move to position 0
      JA Loop

SubW: WAIT POS, 0, 0 //Wait until motor position is at 10000
      WAIT TICKS, 0, 50 //Wait for 50 msec
      RSUB //Return back to main program and execute next line
  
```

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
23	(don't care)	(don't care)	<subroutine address>

**Binary: CSUB SubW** (assuming the address of label 'SubW' is located at address 100)

Byte Index	0	1	2	3	4	5	6	7	8
<b>Function</b>	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
<b>Value (hex)</b>	\$01	\$17	\$00	\$00	\$00	\$00	\$00	\$64	\$7c

## WAIT – Wait for an event to occur

**Description:** Pause the execution of the TMCL program until the specified condition is met. This command is intended for stand-alone operation only and must not be used in direct mode. There are 5 different wait conditions that can be used:

- **TICKS:** wait until the number of timer ticks specified by the <ticks> parameter has been reached. The <motor number> parameter is ignored in this case (set it to zero).
- **POS:** wait until the target position of the motor specified by the <motor> parameter has been reached. An optional timeout value (0 for no timeout) must be specified by the <ticks> parameter.
- **REFSW:** wait until the reference switch of the motor specified by the <motor> parameter has been triggered. An optional timeout value (0 for no timeout) must be specified by the <ticks> parameter.
- **LIMSW:** wait until a limit switch of the motor specified by the <motor> parameter has been triggered. An optional timeout value (0 for no timeout) must be specified by the <ticks> parameter.
- **RFS:** wait until the reference search of the motor specified by the <motor> field has been reached. An optional timeout value (0 for no timeout) must be specified by the <ticks> parameter. The timeout flag (ETO) will be set after a timeout limit has been reached. You can then use a JC ETO command to check for such errors or clear the error using the CLE command.

**Internal function:** The TMCL program counter is held until the specified condition is met.

**Related commands:** JC, CLE

**Syntax:** WAIT <condition>, <motor number>, <ticks>  
 where <condition> is TICKS|POS|REFSW|LIMSW|RFS

**Example:** Wait for motor to reach its position without timing out  
 WAIT POS 0, 0

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
27	0 TICKS – Timer ticks	(don't care)	<# of ticks>
	1 POS – target position reached	0	<# of ticks for timeout> 0 for no timeout
	2 REFSW – reference switch		
	3 LIMSW – limit switch		
	4 RFS – Reference search completed		

### Binary: WAIT POS 0, 0

Byte Index	0	1	2	3	4	5	6	7	8
<b>Function</b>	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
<b>Value (hex)</b>	\$01	\$1b	\$01	\$00	\$00	\$00	\$00	\$00	\$1d

## STOP – Stop the TMCL Program

**Description:** Stops executing a TMCL stand-alone program. This command should be placed at the end of every TMCL program. It is intended for stand-alone operation only and must not be used in direct mode.

**Internal function:** TMCL instruction fetching is stopped.

**Related commands:** none

**Syntax:** STOP

**Example:** End of TMCL Program  
STOP

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
28	(don't care)	(don't care)	(don't care)

### Binary: CSUB SubW (assuming the address of label 'SubW' is located at address 100)

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$1c	\$00	\$00	\$00	\$00	\$00	\$00	\$1d

## SCO – Set Coordinate

**Description:** Up to 20 position values (coordinates) can be stored for use with the MVP COORD command. This command sets a coordinate to a specified value. Note: the coordinate number 0 is only stored in RAM, all are also stored in the EEPROM.

**Related commands:** GCO, CCO, MVP

**Mnemonic:** SCO <coordinate number 0 to 20>, <motor number>, <position>

**Example:** Set coordinate #1 to 1000

```
SCO 1, 0, 1000 //set coordinate #1 to be 1000
MVP COORD, 0, 1 //move to position stored in coordinate #1
```

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
30	<coordinate number>	0	<position>

### Reply in direct mode:

Status	Value
100 – OK	(don't care)

### Binary: SCO 1, 0, 1000

Byte Index	0	1	2	3	4	5	6	7	8
<b>Function</b>	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
<b>Value (hex)</b>	\$01	\$1e	\$01	\$00	\$00	\$00	\$03	\$e8	\$0b

## GCO – Get Coordinate

**Description:** Read a previously stored coordinate. The requested value is copied to the accumulator register for further processing purposes such as conditioned jumps. In direct mode, the value is also output in the value field of the reply. Note: the coordinate number 0 is stored in RAM only, All others are also stored in the EEPROM.

**Internal function:** The desired value is read out of the internal coordinate array, copied to the accumulator register and -in direct mode- returned in the “value” field of the reply.

**Related commands:** SCO, CCO, MVP

**Syntax:** GCO <coordinate number 0 to 20>, <motor number>

**Example:** Get value of coordinate #1

GCO 1, 0

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
31	<coordinate number>	0	(don't care)

### Reply in direct mode:

Status	Value
100 – OK	(don't care)

### Binary: GCO 1, 0

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$1f	\$01	\$00	\$00	\$00	\$00	\$00	\$21

### Reply

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target – address	Target – Address	Status	Instruction	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$02	\$01	\$64	\$0a	\$00	\$00	\$00	\$00	\$71

Value → 0

## CCO – Capture Coordinate

**Description:** The actual positions of the specified axes are copied to the selected coordinate variables. Note: the coordinate number 0 is stored in RAM only, All others are also stored in the EEPROM.

**Internal function:** The selected (24 bit) position values are written to the 20 x 3 bytes wide coordinate array.

**Related commands:** SCO, GCO, MVP

**Syntax:** CCO <coordinate number 0 to 20>, <motor number>

**Example:** Store the current position of motor

CCO 3, 0

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
32	<coordinate number>	0	(don't care)

### Reply in direct mode:

Status	Value
100 – OK	(don't care)

### Binary: CCO 3, 0

Byte Index	0	1	2	3	4	5	6	7	8
<b>Function</b>	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
<b>Value (hex)</b>	\$01	\$20	\$03	\$00	\$00	\$00	\$00	\$00	\$24

## CALCX – Calculate using the X register

**Description:** This instruction is very similar to CALC, but the second operand comes from the X register. The X register can be loaded to the LOAD or the SWAP type of this instruction. The result is written back to the accumulator for further processing like comparisons or data transfer.

**Related commands:** CALC, COMP, JC, AAP, AGP

**Mnemonic:** CALCX <operation>  
with <operation>=ADD|SUB|MUL|DIV|MOD|AND|OR|XOR|NOT|LOAD|SWAP

**Example:** Multiply accu by X-register

CALCX MUL

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
33	0 ADD – add X register to accu 1 SUB – subtract X register from accu 2 MUL – multiply accu by X register 3 DIV – divide accu by X-register 4 MOD – modulo divide accu by x-register 5 AND – logical and accu with X-register 6 OR – logical or accu with X-register 7 XOR – logical exor accu with X-register 8 NOT – logical invert X-register 9 LOAD – load accu to X-register 10 SWAP – swap accu with X-register	0	(don't care)

### Reply in direct mode:

Status	Value
100 – OK	(don't care)

### Binary: CALCX MUL

Byte Index	0	1	2	3	4	5	6	7	8
<b>Function</b>	Target – address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
<b>Value (hex)</b>	\$01	\$21	\$02	\$00	\$00	\$00	\$00	\$00	\$24

## AAP – Accumulator to Axis Parameter

**Description:** The content of the accumulator register is transferred to the specified axis parameter. For practical usage, the accumulator has be loaded e.g. by a preceding GAP instruction. The accumulator may have been modified by the CALC or CALCX (calculate) instruction.

**Related commands:** AGP, SAP, GAP, SGP, GGP, GIO, GCO, CALC, CALCX

**Syntax:** AAP <parameter number>, <motor number>

**Example:** Positioning motor #0 by a potentiometer connected to the analogue input #0:

```

Start: GIO 0, 1      // get value of analogue input line 0
CALC MUL, 4        // multiply by 4
AAP 0, 0           // transfer result to target position of motor 0
JA Start          // jump back to start
    
```

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
34	<parameter number>	0	(don't care)

### Reply in direct mode:

Status	Value
100 – OK	(don't care)

### Binary: AAP 0, 0

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$22	\$00	\$00	\$00	\$00	\$00	\$00	\$23

## AGP – Accumulator to Global Parameter

**Description:** The content of the accumulator register is transferred to the specified global parameter. For practical usage, the accumulator has to be loaded e.g. by a preceding GAP instruction. The accumulator may have been modified by the CALC or CALCX (calculate) instruction. Note that the global parameters in bank 0 are EEPROM only and thus should not be modified automatically by a stand-alone application.

**Related commands:** AAP, SGP, GGP, SAP, GAP, GIO

**Syntax:** AGP <parameter number>, <bank number>

**Example:** Copy accumulator to TMCL user variable #3  
AGP 3, 2

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
35	<parameter number>	<bank number>	(don't care)

### Reply in direct mode:

Status	Value
100 – OK	(don't care)

### Binary: AGP 3, 2

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$23	\$03	\$02	\$00	\$00	\$00	\$00	\$29

## CLE – Clear Error Flags

**Description:** This command clears the internal error flags. It is intended for use in stand-alone mode only and must not be used in direct mode. The following error flags can be cleared by this command (determined by the <flag> parameter):

- ALL: clear all error flags.
- ETO: clear the timeout flag.
- EAL: clear the external alarm flag

**Related commands:** JC

**Syntax:** CLE <flags> where <flags>=ALL|ETO|EDV|EPO

**Example:** Reset the timeout flag  
CLE ETO

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
36	0 – (ALL) all flags 1 – (ETO) timeout flag 2 – (EAL) alarm flag 3 – not in use 4 – not in use 5 – (ESD) shutdown flag	(don't care)	(don't care)

### Binary: CLE ETO

Byte Index	0	1	2	3	4	5	6	7	8
<b>Function</b>	Target – address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
<b>Value (hex)</b>	\$01	\$24	\$01	\$00	\$00	\$00	\$00	\$00	\$26

## User Definable Commands (UF0 to UF7)

**Description:** The user definable functions UF0 through UF7 are predefined, "empty" functions for user specific purposes. See the C-API (application programmer's interface) manual for further details or contact TRINAMIC for customer specific programming of these functions.

**Related commands:** none

**Syntax:** UF0 to UF7

### Binary Representation:

Instruction No.	Type	Motor / Bank	Value
64 to 71	(user defined)	(user defined)	(user defined)

### Reply in direct mode:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target – address	Target – address	Status	Instruction	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$02	\$01	(user defined)	64 to 71	(user defined)	(user defined)	(user defined)	(user defined)	<checksum>

## TMCL Control Functions

**Description:** The following functions are for host control purposes only and are not allowed for stand-alone mode. They are mentioned here only for reasons of completeness. These commands have no mnemonics, as they can not be used in the TMCL assembler. They are to be used only by the TMCL IDE to communicate with the module, for example to download a TMCL application into the module. The only control commands that could be useful for a user host application is "get firmware revision" (command 137, please note the special reply format of this command, described at the end of this section) and eventually 129 (run application). All other functions can be achieved by using the appropriate functions of the TMCL IDE.

Instruction	Description	Type	Motor/Bank	Value
128 – stop application	A running TMCL application is stopped	--	--	--
129 – run application	TMCL execution is started or continued	0 – run from current address 1 – run from specified address	--	Starting address
130 – step application	Only the next command of the TMCL is executed	--	--	--
131 – reset application	Program counter is set to zero, and the standalone application is stopped	--	--	--
132 – start download mode	Target command executed is stopped and all following commands are transferred to the TMCL memory	--	--	Starting address of application
133 – quit download mode	Target command execution is resumed	--	--	--
134 – read TMCL memory	The specified program memory location is read	--	--	<memory address>
135 – get application status	One of these values is returned: 0 – stop 1 – run 2 – step 3 – reset	--	--	--
136 – get firmware revision	Return the module type and firmware revision either as a string or binary	0 – string 1 – binary	--	--
137 – restore factory defaults	Reset all settings stored in the EEPROM to their factory default	--	--	Must be 1234

### Special reply format of command 136:

- Type set to 0: reply as a string:

Byte index	Contents
1	Host Address
2 to 9	Version string (8 characters, e.g. "303V2.48")

There is no checksum in this reply format.

Type set to 1: version number in binary format: Here, the normal reply format is used. The version number is output in the "value" field of the reply in the following way:

Byte index in value field	Contents
1	Version number, low byte
2	Version number, high byte
3	Type number, low byte
4	Type number, high byte

## Axis Parameters

The following sections describe all axis parameters that can be used with the SAP, GAP, AAP, STAP and RSAP commands. Please note that some parameters are different with different module types. The letters under "access" mean: R = readable (GAP), W = writable (SAP), E = automatically restored from EEPROM after reset or power-on.

### Basic Parameters:

Number	Axis Parameter	Description	Range	Access
0	Target position	The desired position in position mode	+/- 2 <sup>23</sup>	RW
1	Actual position	The current position of the motor.	+/- 2 <sup>23</sup>	RW
2	Target speed	The desired speed in velocity mode. In position mode, this parameter is set hardware: to the maximum speed during acceleration and to zero during deceleration.	+/-2047	RW
3	Actual speed	The current rotational speed.	+/-2047	R
4	Max positioning speed	Should not exceed the physically highest possible value. Adjust the pulse divisor (no. 154) if the speed value is very low (<50) or above the upper limit.	0 - 2047	RWE
5	Max acceleration	The limit for acceleration (and deceleration). Changing this parameter requires re-calculation of the acceleration factor (no. 146) and the acceleration divisor (no.137), which is done automatically.	0 - 2047	RWE
6	Absolute max. current	The most important motor setting, since too high values might cause motor damage! The maximum value of 1500 represents a phase current of 1.5 Ampere.	0 - 1500	RWE
7	Standby current	The current limit two seconds after the motor has stopped.	0 - 1500	RWE
8	Target position reached	Indicates that the actual position equals the target position.	0/1	R
9	Reference switch status	The logical state of the reference (left) switch. Default is two switch mode: the left switch as the reference switch, the right switch as a limit (stop) switch.	0/1	R
10	Right limit switch status	The logical state of the (right) limit switch.	0/1	R
11	Left limit switch status	The logical state of the left limit switch (in three switch mode)	0/1	R
12	Right limit switch disabled	if set, deactivates the stop function of the right switch	0/1	RWE
13	Left limit switch disabled	in three switch mode: deactivates the stop function of the left switch if set in one/two switch mode: deactivates the stop function of the reference (left) switch if set	0/1	RWE

### Advanced Parameters:

Number	Parameter	Description	Range	Access
130	Minimum speed	Should always be set 1 to ensure exact reaching of the target position.	0 – 2047	RWE
135	Actual acceleration	The current acceleration. Should never be overwritten.	0 – 2047	R
136	Acceleration threshold	Specifies the threshold between low and high acceleration values for the parameters 144&145. Normally not needed.	0 – 2047	RWE
137	Acceleration divisor	A ramping parameter, can be adjusted in special cases, automatically calculated by setting the maximum acceleration (e.g. during normal initialization).	0 – 13	RWE
138	Ramp mode	Automatically set when using ROR, ROL, MST and MVP. 0: position mode. Steps are generated, when the parameters actual position and target position differ. Trapezoidal speed ramps are provided. 2: velocity mode. The motor will run continuously and the speed will be changed with constant (maximum) acceleration, if the parameter "target speed" is changed. For special purposes, the soft mode (value 1) with exponential decrease of speed can be selected.	0/1/2	RWE
139	Interrupt flags	Must not be modified.	16 bits	RW

Number	Parameter	Description	Range	Access
140	Microstep resolution	0 – full step 1 – half step 2 – 4 microsteps 3 – 8 microsteps 4 – 16 microsteps 5 – 32 microsteps 6 – 64 microsteps Modifying this parameter will affect the rotation speed.	0 to 6	RWE
141	Ref. switch tolerance	For three-switch mode: a position range, where an additional switch (connected to the REFL input) won't cause motor stop.	0 – 4095	RW
142	Snapshot position	For referencing purposes, the exact position at hitting of the reference switch can be captured in this parameter. A dummy value has to be written first to prepare caption.	+/- 2 <sup>23</sup>	RW
143	Max. current at rest	In contrast to the standby current, this current limit becomes immediately active when the motor speed reaches zero. The value represents a fraction of the absolute maximum current: 0 – no change of current at rest (default, 100%) 1 – 12.5% 2 – 25% 3 – 37.5% 4 – 50% 5 – 62.5% 6 – 75% 7 – 87.5%	0 – 7	RWE
146	Acceleration factor	A ramping parameter, can be adjusted in special cases, automatically calculated by setting the maximum acceleration (normal initialization).	0 – 128	RWE
147	Ref. switch disable flag	If set, the reference switch (left switch) won't cause the motor to stop. See parameters 12 and 13.	0/1	RWE
148	Limit switch disable flag	If set, the limit switch (right switch) won't cause the motor to stop. See parameters 12 and 13.	0/1	RWE
149	Soft stop flag*	If cleared, the motor will stop immediately (disregarding motor limits), when the reference or limit switch is hit.	0/1	RWE
150	Reserved	Do not modify	0/1	R
151	Position latch flag	Indicates that a position snapshot has been completed (see parameter 142)	0/1	R
152	Interrupt mask	Must not be modified. See the TMC 428 datasheet for details.	16 bits	R
153	Ramp divisor	The <b>exponent</b> of the scaling factor for the ramp generator should be de/incremented carefully (in steps of one).	0 – 15	RWE
154	Pulse divisor	The <b>exponent</b> of the scaling factor for the pulse (step) generator – should be de/incremented carefully (in steps of one).	0 – 15	RWE
193	Reference mode	Specifies the number of switches for the reference search.	1/2/3	RWE
194	Referencing search speed	For the reference search this value specifies the search speed as a fraction of the maximum velocity: 0 – full speed 1 – half of the maximum speed 2 – a quarter of the maximum speed 3 – 1/8 of the maximum speed (etc.)	0 – 8	RWE
195	Referencing switch speed	Similar to parameter no. 194, the speed for the switching point calibration can be selected.		RWE
196	Reserved	Must not be changed		
197	Steps per cycle	A value unequal to zero starts the cyclic mode: according to this value, the parameters "cyclic position" and "turn counter" are calculated (Versions 2.22 and later). If microstepping is used, the number microsteps per cycle must be set.	0 – 2 <sup>15</sup>	RWE
203	Mixed decay threshold	If the actual velocity is above this threshold, mixed decay will be used. This can also be set to –1 which turns on mixed decay permanently also in the <b>rising part</b> of the microstep wave. This can be used to fix microstep errors.	0 – 2048 or -1	RWE
204	Freewheeling	Time after which the power to the motor will be cut when its velocity has reached zero.	0 – 65335 0 = never	RWE
205	Stall detection threshold	Stall detection threshold. Set it to 0 for no stall detection or to a value between 1 (low threshold) and 7 (high threshold). The motor will be stopped if the load value exceeds the stall detection threshold.	0 – 7	RWE
206	Actual load value	Readout of the actual load value used for stall detection.	0 – 7	R

## Global Parameters

Global parameters are grouped into 3 banks: bank 0 (global configuration of the module), bank 1 (user C variables) and bank 2 (user TMCL variables). The letters under "access" mean: R = readable (GGP), W = writeable (SGP), E = automatically restored from EEPROM after reset or power-on. Use SGP and GGP commands to write and read global parameters.

### Bank 0

It is best to set these parameters by using the appropriate functions of the TMCL DIE and not by entering many SGP commands (the TMCL IDE does this automatically).

Number	Parameter
0	Datagram low word (read only)
1	Datagram high word (read only)
2	Cover datagram position
3	Cover datagram length
4	Cover datagram content
5	Reference switch states (read only)
6	TMC428 SMGP register
7 – 22	Driver chain configuration long words 0 – 15
23 – 32	Microstep table long word 0 – 15

An STGP 23, 0 command will store the entire microstep table, and an STGP 7, 0 command will store the entire driver chain configuration table. **Use the appropriate functions of the TMCL IDE to change these parameters interactively, if really necessary! Take extreme care when doing this, as wrong configurations here may cause damage to the motor drivers!**

The following parameters with the numbers from 64 on configure things like the serial address of the module RS232 / RS485 baud rate or CAN bit rate. Change these parameters to meet your needs. The best and easiest way to do this is to use the appropriate functions of the TMCL IDE. The parameters with numbers between 64 and 128 are stored in EEPROM only, so that an SGP command on such a parameter will always store it permanently (no extra STGP command needed). **Take care when changing these parameters, and use the appropriate functions of the TMCL IDE to do it in an interactive way.**

Number	Parameter	Description	Range	Access
64	EEPROM	Setting this parameter to a different value as \$E4 will cause re-initialization of the axis and global parameters (to factory defaults) after the next power up. This is useful in case of miss-configuration.	0 – 255	RWE
65	RS485 Baud Rate	0 – 9600 baud (default)    1 – 14400 baud 2 – 19200 baud            3 – 28800 baud 4 – 38400 baud            5 – 57600 baud 6 – 76800 baud            7 – 115200 baud	0 – 7	RWE
66	Serial address	The module (target) address for RS485	0 – 255	RWE
72	System error mask	1 – 10kBits/s    2 – 20oBits/s 3 – 50kBits/s    4 – 100kBits/s 5 – 125kBits/s   6 – 250kBits/s (default) 7 – 500kBits/s   8 – 1000kBits/s	1 – 7	RWE
73	Configuration EEPROM lock flag	Write: 1234 to lock the EEPROM, 4321 to unlock it. Read: 1 = EEPROM locked, 0 = EEPROM unlocked.	0/1	RWE
74	Broadcast mask	(currently not used, do not change)		RWE
75	Telegram pause time	Pause time before the reply via RS485 will be sent. Set it to 15 for RS485 adapters controlled by the RTS pin)	0 – 255	RWE
76	Serial host address	Host address used in the reply telegrams sent back via RS485	0 – 255	RWE
77	Auto start mode	0: Do not start TMCL application after power-up (default). 1: Start TMCL application automatically after power-up.	0/1	RWE
78	Poll interval	Internal polling rate of motor speeds and switch states. 0	0 – 255	RWE

		corresponds to an interval of 166 microseconds, 255 results in a rate of 42.6 milliseconds (256 * 166us) default is 12 (2ms).		
Number	Parameter	Description	Range	Access
79	Port function mask	Alternate functions of the port pins (ref. switch status, target position reached).	8 bits	RWE
80	Shutdown pin functionality	Select the functionality of the SHUTDOWN pin: 0 – no function, 1 – high active, 2 – low active.	0 – 2	RWE
128	TMCL application status	0 – stop    1 – run 2 – step    3 – reset	0 – 3	R
129	Download mode	0 – normal mode    1 – download mode	0/1	R
130	TMCL program counter	The index of the currently executed TMCL instruction		R

### Bank 1

The global parameter bank 1 contains a set of variables that are mainly intended for use in customer specific extensions to the firmware. So, together with the user definable commands these variables form the interface between extensions to the firmware (written in C) and a TMCL application. Although they could also be used as general purpose variables in TMCL programs, it is much better to use the variables in bank 2 for this purpose.

Number	Global Parameter	Description	Range	Access
0	C application state	The main state machine variable of the example user C code.	0 – 255	RW
1	(not used)		0 – 255	RW
2	C application state timer	A universal timer, supposed for state timing purposes.	0 – 255	RW
3	C application general purpose variable "unsigned char #0"		0 – 255	RWE
4	C application general purpose variable "unsigned char #1"		0 – 255	RWE
5	C application general purpose variable "unsigned char #2"		0 – 255	RWE
6	C application general purpose variable "unsigned int #0"		0 – 2 <sup>16</sup>	RWE
7	C application general purpose variable "unsigned int #1"		0 – 2 <sup>16</sup>	RWE
8	C application general purpose variable "unsigned int #2"		0 – 2 <sup>16</sup>	RWE
9	C application general purpose variable "signed long #0"		-2 <sup>31</sup> to +2 <sup>31</sup>	RWE
10	C application general purpose variable "signed long #1"		-2 <sup>31</sup> to +2 <sup>31</sup>	RWE
11	C application general purpose variable "signed long #2"		-2 <sup>31</sup> to +2 <sup>31</sup>	RWE

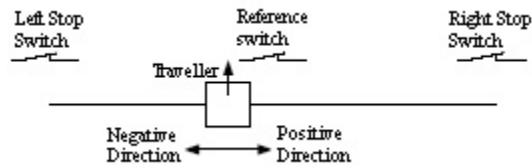
## Bank 2

Bank 2 contains general purpose 32 bit variables for the use in TMCL applications. They are located in RAM and can be stored to EEPROM. After booting, their values are automatically restored to the RAM. By default, 20 variables are available.

Number	Global Parameter	Description	Range	Access
0	General purpose variable #0	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
1	General purpose variable #1	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
2	General purpose variable #2	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
3	General purpose variable #3	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
4	General purpose variable #4	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
5	General purpose variable #5	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
6	General purpose variable #6	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
7	General purpose variable #7	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
8	General purpose variable #8	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
9	General purpose variable #9	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
10	General purpose variable #10	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
11	General purpose variable #11	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
12	General purpose variable #12	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
13	General purpose variable #13	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
14	General purpose variable #14	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
15	General purpose variable #15	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
16	General purpose variable #16	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
17	General purpose variable #17	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
18	General purpose variable #18	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE
19	General purpose variable #19	For use in TMCL applications	$-2^{31}$ to $+2^{31}$	RWE

## Reference Searching

The built-in reference search features switching point calibration and support of one or two reference switches. The internal operation is based on three individual state machines (one per axis) that can be started, stopped and monitored (instruction RFS, no. 13). The settings of the automatic stop functions corresponding to the switches (axis parameters 12 and 13) have no influence on the reference search.



*Definition of switches*

Selecting the referencing mode (axis parameter 193): in modes 1 and 2, the motor will start by moving "left" (negative position counts). In mode 3 (three-switch mode), the right stop switch is searched first to distinguish the left stop switch from the reference switch by the order of activation when moving left (reference switch and left limit switch share the same electrical function). In circular mode (axis parameter 197 greater than zero), mode 1 (one switch) configuration is assumed, ignoring parameter 193.

- Until the reference switch is found for the first time, the searching speed is identical to the maximum positioning speed (parameter 4), unless reduced by axis parameter 194.
- After hitting the reference switch, the motor slowly moves right until the switch is released. Finally the switch is re-entered in left direction, setting the reference point to

the center of the two switching points. This low calibrating speed is a quarter of the maximum positioning speed by default (axis parameter 195).

## Stall Detection

Stall detection means that the motor will be stopped when the load gets too high. It is controlled by axis parameter #205. If this parameter is set to a value between 1 and 7 the stall detection will be activated. Setting it to 0 means that stall detection is turned off. A greater value means a higher threshold. This also depends on the motor and on the velocity. There is no stall detection while the motor is being accelerated or decelerated. Stall detection can also be used for finding the reference point. You can do this by using the following TMCL code:

```
SAP 205, 0, 5 //Turn on Stall Detection (use other threshold if needed)
ROL 0, 500 //Let the motor run (or use ROR or other velocity)
Loop: GAP 3, 0
      COMP 0
      JC NE, Loop //Wait until the motor has stopped
      SAP 1, 0, 0 //Set this position as the zero position
```

Do not use RFS in this case.

## Fixing microstep errors

Due to the “zero crossing problem” of all stepper motor drivers, microstep errors may occur with some motors as the minimum motor current that can be reached is slightly higher than zero (depending on the inductivity, resistance and supply voltage of the motor). This can be solved by setting the “mixed decay threshold” parameter (axis parameter number 203) to the value -1. This switches on mixed decay permanently, in every part of the microstepping waveform. Now the minimum reachable motor current is always near zero which gives better microstepping results. A further optimization is possible by adapting the motor current shape. Use SAP 203, 0, -1 to turn on this feature.

## The syntax of TMCL in the TMCL assembler

Here, the syntax of the TMCL commands used by the TMCL assembler is given. The command mnemonics given there are used in the TMCL assembler.

### Assembler directives

Assembler directives always start with a # sign. The only directive is *#include* to include a file. The name of that file must be given after the *#include* directive. If that file is already in the editor, it will be taken from there. Otherwise it will be loaded from file, using the include file path that can be set in the “Options” dialogue. Example: *#include test.tmc*

### Symbolic constants

Symbolic constants are defined using the syntax *<Name>=<Value>*. A name must always start with a letter or the sign *\_* and may then contain any combination of letters, numbers and the sign *\_*. A value must always be a decimal, hexadecimal or binary number or a constant expression (see section 7.7.3). Hexadecimal numbers start with a \$ sign, binary numbers start with a % sign. Examples:

```
Speed=1000
Speed2=Speed/2
Mask=$FF
BinaryValue=%1010101
```

## Constant expressions

Wherever a numerical value is needed, it can also be calculated during assembly. For this purpose constant expressions can be used. A constant expression is just a formula that evaluates to a constant value. The syntax is very similar to BASIC or other programming languages. Please note that the calculation takes place during assembly and not during execution of the TMCL program on the module. Internally, the assembler uses floating point arithmetic to evaluate a constant expression, but as TMCL commands only take integer values, the result of a constant expression will always be rounded to an integer value when used as an argument to a TMCL command. Here is a list of functions and operators that can be used in constant expressions:

- Functions:

Name	Function
SIN	Sine
COS	Cosine
TAN	Tangent
ASIN	Arc sine
ACOS	Arc cosine
ATAN	Arc tangent
LOG	Logarithmic Base 10
LN	Logarithmic Base e
EXP	Exponential power to the base e
SQRT	Square root
ABS	Absolute value
INT	Integer (truncate)
ROUND	Integer (round)
SIGN	Returns a: -1 if argument < 1 0 if argument = 0 1 if argument > 1
DEG	Converts from radians to degrees
RAD	Converts from degrees to radians

- Operators:

Symbol	Meaning
()	Parenthesis
^	Power
*	Multiplication
/	Division
+	Addition
-	Subtraction

Symbolic constants, floating point numbers, integer numbers, hexadecimal numbers and binary numbers can also be used in constant expressions. Here are some examples of constant expressions used wherever constant values can be placed:

```
ROL 0, 7+9*8
Speed2=Speed*SIN(0.5)
MVP ABS, 0, 3*1000
Sin90=Sin(Rad(90))
```

## Labels

Example (the label has the name "Loop"):

```
Loop: MVP ABS, 0, 1000
      WAIT POS, 0, 0
      MVP ABS, 0, 0
      WAIT POS, 0, 0
      JA Loop
```

## Comments

Comments always start with // (like in C++). The rest of the line is then ignored.

## TMCL Commands

Here is a list of all command mnemonics that are recognized by the assembler.

ROR <n1>, <n2>  
ROL <n1>, <n2>  
MST <n1>  
MVP <mvp\_opt>, <n1>, <n2>  
SAP <n1>, <n2>, <n3>  
GAP <n1>, <n2>  
STAP <n1>, <n2>  
RSAP <n1>, <n2>  
SGP <n1>, <n2>, <n3>  
GGP <n1>, <n2>  
STGP <n1>, <n2>  
RSGP <n1>, <n2>  
RFS <rfs\_opt>, <n1>  
SIO <n1>, <n2>, <n3>  
GIO <n1>, <n2>  
CALC <op1>, <n2>, <n3>  
CALCX <op2>, <n2>, <n3>  
COMP <n1>  
JC <cc>, <Label>  
JA <Label>  
CSUB <Label>  
RSUB  
WAIT <Event>, <n1>, <n2>  
STOP  
SAC <n1>, <n2>  
SCO <n1>, <n2>, <n3>  
GCO <n1>, <n2>  
CCO <n1>, <n2>  
AAP <n1>, <n2>  
AGP <n1>, <n2>  
CLE <Flag>

with:

<n1>, <n2>, <n3>: Any numerical value, constant expression or symbolic constant

<mvp\_opt>: An option for MVP: ABS, REL or COORD.

<rfs\_opt>: An option for RFS: START, STOP or STATUS.

<cc>: A condition code: ZE, NZ, EQ, NE, GT, GE, LT, LE, ETO, EAL, EDV, EPO.

<Event>: A wait event. This can be TICKS, POS, LIMSW, REFSW or RFS.

<op1>: An operator for the CALC command: ADD, SUB, MUL, DIV, MOD, AND, OR, NOT, LOAD

<op2>: An operator for the CALCX command: ADD, SUB, MUL, DIV, MOD, AND, OR, NOT, LOAD, SWAP.

<Label>: A label defined somewhere else in the program.

<Flag>: An error flag code: ALL, ETO, EAL, EDV or EPO.

## Sample code

### General Parameters

```
//TMCL Sample Setup Program
StallValue = 0 // StallGuard Threshold (0 = off)
Velocity = 2047 // Max Velocity
Acceleration = 1100 // Max Acceleration
ustep = 4 // 4 = 16X u-stepping
pdiv = 3 // pdiv @ default
rdiv = 8 // rdiv
current = 600 // 600 mA Peak
Hcurrent = 300 // 300 mA Peak
decay = 0 // 2048 = on, 0 = off

//Initialization of above parameters
SAP 6, 0, current //Setting Current
SAP 203, 0, decay //Switch Mixed Decay
SAP 5, 0, Acceleration //Setting max acceleration
SAP 205, 0, StallValue //Setting StallGuard Threshold
SAP 140, 0, ustep //Setting u-stepping
SAP 143, 0, Hcurrent //Max Hold Current
SAP 154, 0, pdiv //Initializing pdiv
SAP 4, 0, Velocity //Setting maximum Velocity
SAP 138, 0, rdiv //Ramp Divisor
ROR 0, 1049 //Rotate Right Continuously
WAIT TICKS, 0, 1000 //Wait for 1000 msec before issuing next command
MST 0 //Stop Motor # 0
STOP //End program
```

### Potentiometer Control #1 – Run continuously

```
//TMCL POT Control for TMCM-110. This will move step motor
//up to top speed when pot is turned to maximum, when pot is 0, motor is 0 speed

//Parameters
StallValue = 0 //StallGuard Threshold (0 = off)
Velocity = 2047 //Max Velocity
Acceleration = 1000 //Max Acceleration
ustep = 1 //4 = 16X u-stepping
pdiv = 8 //pdiv @ default
rdiv = 8 //rdiv
current = 600 //600 mA Peak
Hcurrent = 300 //300 mA Peak
decay = 2048 //2048 = on, 0 = off
speed = 500 //Rotation Speed of ROR Command

//Connect pot: 1 side to +5V output, pin 8. Other side to GND, pin 6,
//and the middle wire to the input line, pin 7. Pot resistance acts as a voltage divider

//Initialization of above parameters
SAP 6, 0, current //Setting Current
SAP 203, 0, decay //Switch Mixed Decay
SAP 5, 0, Acceleration //Setting max acceleration
SAP 205, 0, StallValue //Setting StallGuard Threshold
SAP 140, 0, ustep //Setting u-stepping
SAP 143, 0, Hcurrent //Max Hold Current
SAP 153, 0, rdiv //Initializing rdiv
SAP 154, 0, pdiv //Initializing pdiv
```

```

SAP 4, 0, Velocity      //Setting maximum Velocity
SAP 138, 0, 2          //Ramp Mode

//Main
main:
  GIO 0, 1              //read pot
  CALC MUL, 17          //Run calculations to make it a valid number
  CALC DIV, 10
  WAIT TICKS, 0, 5     //Small Delay
  AAP 132, 0           //transfers result to target position
  JA main              //Infinite Loop
  STOP                 //End of Program

```

## Potentiometer Control #2 – CW and CCW rotation

```

//TMCL POT Control for TMCM-110. This will move step motor
// when pot is 0, motor is rotating CW, when pot is half way, motor is stopped,
// when pot is max, motor is rotating CCW (depending on connection of motor)

```

```

StallValue = 0        //StallGuard Threshold (0 = off)
Velocity = 2047       //Max Velocity
Acceleration = 1000   //Max Acceleration
ustep = 1             //4 = 16X u-stepping
pdiv = 8              //pdiv @ default
rdiv = 8              //rdiv
current = 600         //600 mA Peak
Hcurrent = 300        //300 mA Peak
decay = 2048         //2048 = on, 0 = off

```

```

//Connect pot: 1 side to +5V output, pin 8. Other side to GND, pin 6,
// and the middle wire to the input line, pin 7. Pot resistance acts as a voltage divider

```

```

//Initialization of above parameters
SAP 6, 0, current     //Setting Current
SAP 203, 0, decay     //Switch Mixed Decay
SAP 5, 0, Acceleration //Setting max acceleration
SAP 205, 0, StallValue //Setting StallGuard Threshold
SAP 140, 0, ustep     //Setting u-stepping
SAP 143, 0, Hcurrent  //Max Hold Current
SAP 154, 0, pdiv      //Initializing pdiv
SAP 4, 0, Velocity    //Setting maximum Velocity

```

```

LOOP:
  GIO 0, 1            // get status of analog input
  COMP 0              // compare to value 0
  JC EQ, GORIGHT     // rotate right if equal
  COMP 1015          // compare to value 1015 (1023 is highest)
  JC GE, GOLEFT      // rotate left if greater or equal to 1015
  JC LT, STOPMOTOR   // stop motor if less than 1015
  JA LOOP            // loop continuously
  STOP

```

```

GORIGHT:              //Rotate right if pot is at min
  ROR 0, 1000
  GAP 1, 0
  COMP 1000
  JC GE, STOPMOTOR
  JA LOOP

```

```

GOLEFT:                                //Rotate left if pot is at max
    ROL 0, 1000
    GAP 1, 0
    COMP 0
    JC LE, STOPMOTOR
    JA LOOP

STOPMOTOR:                              //Stop rotation if pot is in the middle
    MST 0
    JA LOOP

```

### Potentiometer Control #3 – Position control

//TMCL POT Control for TMCM-110. This will move motor 2400 steps if pot is 0,  
// and move back to position 0 (back by 10000 steps) if pot is max. It will not  
// rotate if pot is in middle

```

//Parameters
StallValue = 0           //StallGuard Threshold (0 = off)
Velocity = 2047         //Max Velocity
Acceleration = 1000     //Max Acceleration
ustep = 1               //4 = 16X u-stepping
pdiv = 8                //pdiv @ default
rdiv = 8                //rdiv
current = 600           //600 mA Peak
Hcurrent = 300          //300 mA Peak
decay = 2048            //2048 = on, 0 = off

```

//Connect pot: 1 side to +5V output, pin 8. Other side to GND, pin 6,  
// and the middle wire to the input line, pin 7. Pot resistance acts as a voltage divider

```

LOOP:  GIO 0, 1          // get status of analog input
    COMP 0               // compare to value 0
    JC EQ, CLOSE_VALVE  // rotate right if equal
    COMP 500            // compare to value 500
    JC GE, OPEN_VALVE   // rotate left if greater or equal to 1015
    JC LT, STOPMOTOR    // stop motor if less than 1015
    JA LOOP              // loop continuously
    STOP

```

```

OPEN_VALVE:
    MVP ABS, 0, 10000
    JA LOOP

```

```

CLOSE_VALVE:
    MVP ABS, 0, 0
    JA LOOP

```

```

STOPMOTOR:
    MST 0
    JA LOOP

```

### Left and Right Limit Switches

//TMCL Sample Setup Program. If no limit switches are tied to ground, no motion will occur.  
//If the left limit switch is tied to ground, it will rotate left until the left limit switch goes high.  
//If the right limit switch is tied to ground, it will rotate right until it goes high.

```

//Parameters
StallValue = 0           //StallGuard Threshold (0 = off)
Velocity = 2047          //Max Velocity
Acceleration = 1000      //Max Acceleration
ustep = 1                //4 = 16X u-stepping
pdiv = 8                 //pdiv @ default
rdiv = 8                 //rdiv
current = 600            //600 mA Peak
Hcurrent = 300           //300 mA Peak
decay = 2048             //2048 = on, 0 = off

//Connect a 3 position switch such that the Left Limit switch (Pin 1) is connected to one side
//of the switch, the Right Limit switch (Pin2) is connect to the other side of the switch,
//and Ground (Pin 3) is in the middle position of the switch.

//Initialization of above parameters
SAP 24, 0, decel         //Set deceleration
SAP 6, 0, current        //Setting Current
SAP 203, 0, decay        //Switch Mixed Decay
SAP 5, 0, Acceleration   //Setting max acceleration
SAP 205, 0, StallValue   //Setting StallGuard Threshold
SAP 140, 0, ustep        //Setting u-stepping
SAP 143, 0, Hcurrent     //Max Hold Current
SAP 154, 0, pdiv         //Initializing pdiv
SAP 4, 0, Velocity       //Setting maximum Velocity
SAP 12, 0, 1             //activate right switch
SAP 13, 0, 1             //activate left switch

Loop:
CSUB CheckLeftLim        //Call subroutine
CSUB CheckRightLim       //Call subroutine
JA Loop
STOP

CheckLeftLim:
GAP 11, 0                 //Get status of Left Limit switch
COMP 0
JC EQ, GoMotorLeft       //If equal to zero, rotate
COMP 1
JC EQ, StopMotor         //If equal to one, stop motor
RSUB

CheckRightLim:
GAP 10, 0
COMP 0
JC EQ, GoMotorRight      //If equal to zero, rotate
COMP 1
JC EQ, StopMotor         //If equal to one, stop motor
RSUB

StopMotor:
MST 0
RSUB

GoMotorRight:
ROR 0, 1000
JA CheckRightLim

GoMotorLeft:
ROL 0, 1000
JA CheckLeftLim

```