Dynamic Motor Motion
Technology Corporation

DYN2 Series
AC Servo Drive Specification

TYPE A - GENERAL PURPOSE PULSE / ANALOG / RS232
TYPE B - MODBUS
TYPE C - CAN

Manual Code : DYN2MS-14F-0917A18	Hardware Version : AH10
Revision : A1.8	Firmware Version : AH10

■ Safety Notice ■

The user or operator should read through this manual completely before installation, testing, operation, or inspection of the equipment. The DYN2 series AC Servo Drive should be operated under correct circumstances and conditions. Bodily harm or damage to equipment and system may result if specifications outlined in this document are not followed. Take extra precaution when the warning convention is used.

⚠ WARNING

■ Notations Used ■

All specification and units of measurement used in the manual are in METRIC:
>   Mass: Kilogram [kg]
>   Length: Millimeter [mm]
>   Time: Seconds [s]
>   Temperature: Celsius [°C]

■ Standards Compliance ■

CE	Machinery Directive 2006/42/EC Low Voltage Directive 2014/35/EU Electromagnetic Compatibility 2014/30/EU EN12100:2010 EN 60034-1:2010; EN 60204-1:2006/AC: 2010 EN 61000-6-1: 2007; EN 61000-6-2:2005/AC: 2005

Product Manual Preface

The user or operator should read through this manual completely before installation, testing, operation, or inspection of the equipment. The DYN2 series AC Servo Drive should be operated under correct circumstances and conditions. Bodily harm or damage to equipment and system may result if specifications outlined in this document are not followed. Take extra caution at details when the warning convention is used.

This manual is available on the DMM Technology Corp. website. A physical copy or reference to the on-line availability must be kept convenient to the servo drive user or operator for references. Contact DMM Technology Corp. if the user or operator has any questions or concerns regarding use.

The DYN2 AC Servo Drive is not designed or certified to implement safety into a system and should not be used in applications where the servo drive maintains safety to personnel or machine.

To reflect improvements, additions, revisions, changes or corrections made to the product or manual, this document is updated accordingly and divided into revisions to reflect each version.

Manual Contents

A.1   Introduction

This manual documents all features and specifications for the DYN2 series AC Servo Drive Type A - General Purpose Pulse/Analog. The servo drive features standard pulse train and analog command input modes compatible with universal motion controllers, PLC's or CNC controllers. Control modes include position, speed or torque servo mode with standard signal connections and interfacing for seamless integration into any system. A high resolution 16-bit (65,536pulse/rev) encoder combined with outstanding 10ms instantaneous position response optimizes performance in high-demand applications.

Gain adjustment is simplified with 3 parameter tuning for fast and easy adjustments while maintaining critical application and response flexibility. All testing and tuning is done through a RS232 or USB interface with a host PC running DMMDRV GUI software for fast and easy set up. Drive status is internally monitored by 22 parameters for consistent and reliable performance.

Standard servo motor and encoder/motor power cable pair options available. Measuring only 32mm [W] x 85mm [H] x 75mm [D], the DYN2 AC Servo drive can power up to 0.75kW (7.1Nm) capacity. The perfect servo drive for any small to medium capacity application.

A.2   Name Plate

Note the name plate is region specific and may vary between each region model.

DYN2 - T L A 6 S - 00

DMM Servo Drive Series

DYN2 AC Servo Drive Low Voltage DC Input	

Voltage Class

T	60V

Encoder

4	14-bit
6	16-bit

Model Type

S	Standard
**	Custom

Motor Capacity

1	50W - 200W
L	400W - 750W

Command Type

A	Pulse / Analog / RS232 (DYN232M Protocol)
B	Modbus RTU (RS485)
C	CAN (DMM Proprietary CAN Protocol)
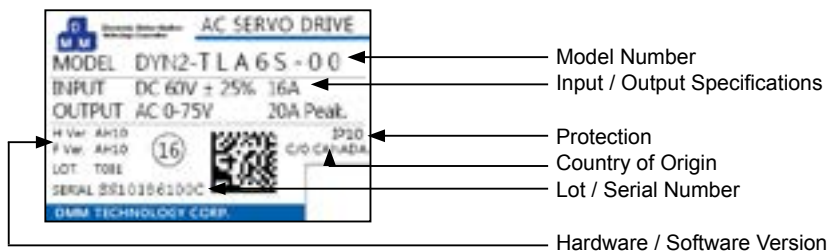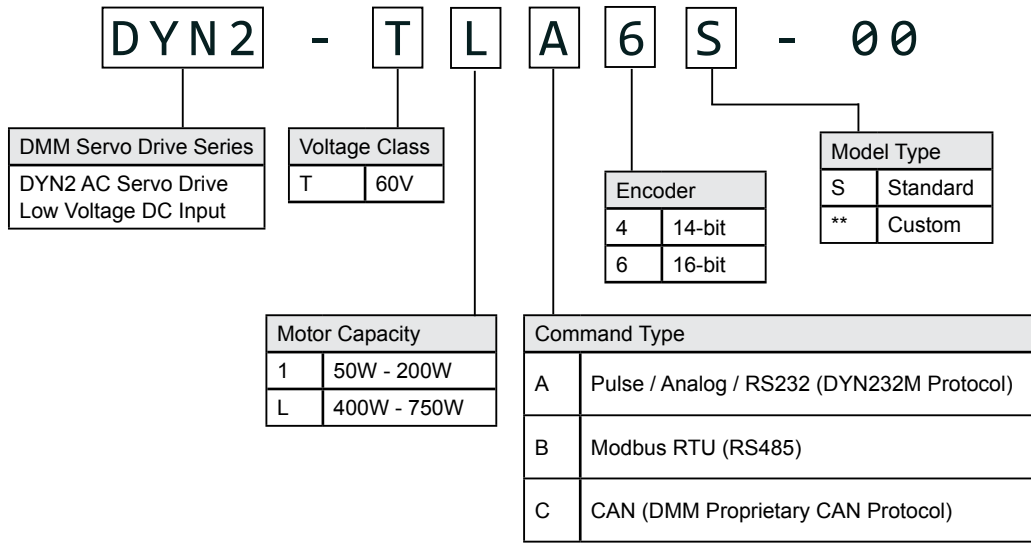
1   GENERAL SPECIFICATION

1.1 Drive Overall Specification

	Data	Specification
Input	Rated Voltage	60VDC ± 10%
	Permissible Input Voltage	24VDC ~ 75VDC
	Rated Current	16A
Output	Rated Voltage	Peak. +75VAC Between any two motor phase
	Rated Current	[L] Capacity Model: Peak. 20A [1] Capacity Model: Peak. 10A From any single motor phase
	Motor Capacity	50W ~ 750W
Drive Interface Power Supply (JP2 Pin. 12)	Voltage	5VDC +/-%5
	Max. Current Draw	50mA
Control Method		SVPWM
Dynamic Brake		Integrated
Encoder Feedback		14/16-bit Single-Turn Absolute
Protection Functions		Current, Voltage, Temperature, Over Power, Position Lost Follow
Position Servo	Command Reference Pulse*1	Pulse+Sign, A/B Phase Quadrature 90° Phase Differential, CW+CCW
	Max. Input Frequency	500kHz
	Input Voltage	5VDC ± %5 (Higher voltage available as option) Over drive photocoupler diode
	Positioning Feedback	Z Index Pulse*2
Speed Servo	Speed Control Range	0:5000
	Input Reference Voltage	-10VDC ~ +10VDC ± %5 3,000rpm at ± 5VDC
	Max Input Voltage	± 12VDC
Torque Servo	Input Reference Voltage	-10VDC ~ +10VDC ± %5
	Max Input Voltage	± 12VDC
Environment	Protection	IP10
	Operation Temperature	0~55°C
	Storage Temperature	-20 ~ 65°C
	Max. Operation Humidity	95RH% (no dew)
	Max. Storage Humidity	95RH% (no dew)
Mass		0.2kg

1. CW+CCW command format available as option.
2. See section 4.5 for Z index pulse details

1.2 Control Block Diagram
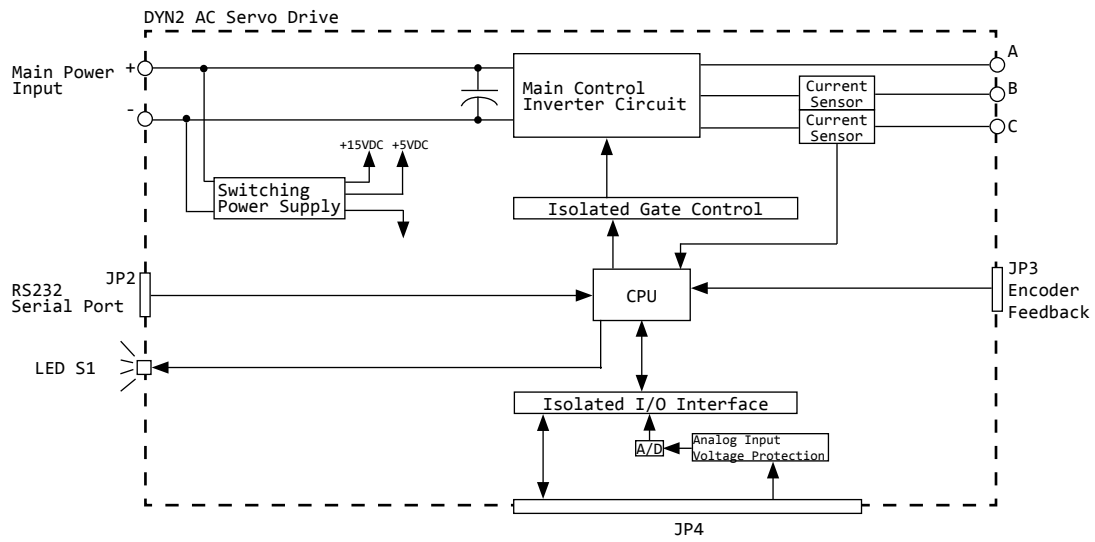
1.3 Encoder Specification

■ Model

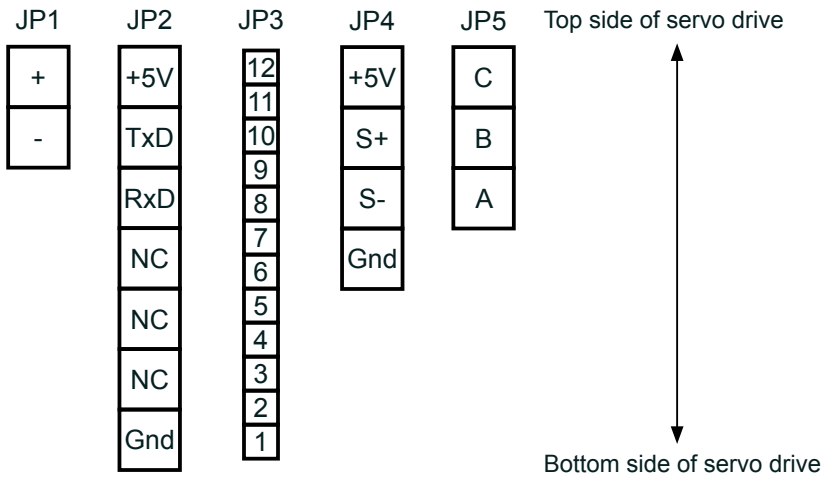| Model Number	Type	Resolution	Data Type	Interface Type	Measurement	Voltage	Status
ABS-14-00	Absolute	14bit [16,384ppr]	6-Wire Serial	Differential Driver/ Receiver	Magnetic	+5VDC	A
ABS-16-00	Absolute	16bit [65,536ppr]	4-Wire Serial	Differential Driver/ Receiver	Magnetic	+5VDC	A

DYN2MS-02F-0116A17

2.1 DYN2 Servo Drive Body Layout

■ JP1
Power Supply Input

(1)

■ JP5
Servo Motor Power

■ JP2
RS232 Port to PC

■ JP3
I/O Port

■ S1 Drive Status LED
[Green LED]

■ JP4
Encoder Feedback Port

(1) Drive Body Grounding Terminal
[M3 x 5mm Max.]
*Located on left side of body
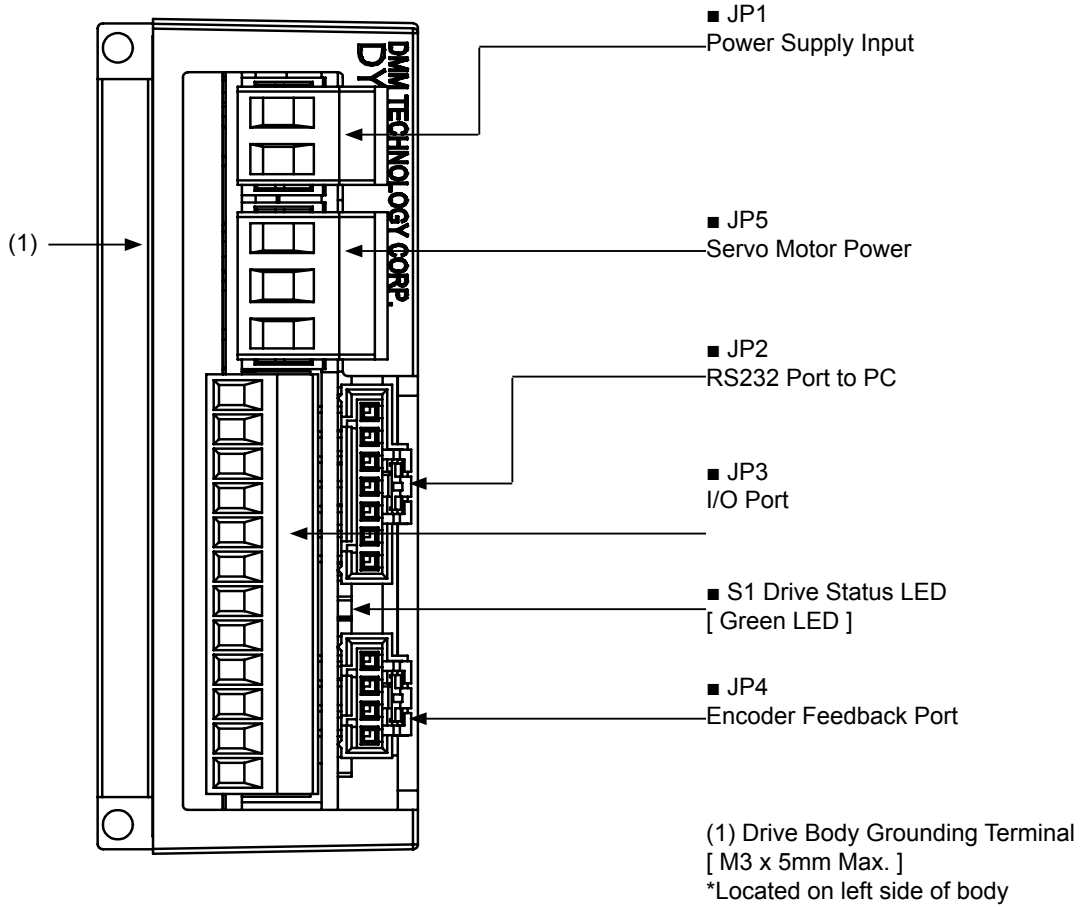
■ Pin Layout

DO NOT reverse JP1 DC input polarity. Reversing polarity will permanently damage servo drive.

JP1	JP2	JP3	JP4	JP5
+	+5V	12	+5V	C
		11		
-	TxD	10	S+	B
		9		
	RxD	8	S-	A
		7		
	NC	6	Gnd	
		5		
	NC	4		
		3		
	NC	2		
	Gnd	1		

Top side of servo drive

Bottom side of servo drive

2.2 Connector and Signal Specification

▌ JP1 - Power Supply Input

Connector Type: 5.00mm Pitch Terminal Block

Drive Header: Phoenix MSTBA 2,5/ 2-G

Plug Connector: Phoenix MSTB 2,5/ 2-ST

Recommended Wire Gauge: 0.8mm^2 (AWG18)

▌ JP2 RS232 Port to PC

Connector Type: 2.54mm Pitch Rectangular

Drive Header: Molex 70553-0041

Plug Connector: Molex 50-57-9407

Recommended Wire Gauge: 0.3mm^2 (AWG22)

Signal Layout:

Type A	Type B (Modbus RS485)	Type C (CAN)
Pin 1: GND	Pin 1: GND	Pin 1: GND
Pin 2~4: NC	Pin 2: RS485+	Pin 2: CANH
Pin 5: RS232 signal input, RxD, TTL/CMOS level.	Pin 3: RS485-	Pin 3: CANL
Pin 6: RS232 signal output, TxD, TTL/CMOS level.	Pin 4: NC	Pin 4: NC
Pin 7: +5(V) output, <10(mA), generated in board.	Pin 5: RS232 RxD	Pin 5: RS232 RxD
	Pin 6: RS232 TxD	Pin 6: RS232 TxD
	Pin 7: +5V	Pin 7: +5V

In order to connect JP2 with PC's RS232 port, an intermediate cable with level shift buffer is necessary. Intermediate cable shipped with drive tuning cable [Model No. CA-MRS232-6].

▌ JP3 I/O Port - Position Command Input

Connector Type: 3.5mm Pitch Terminal Block

Drive Header: Phoenix MC 1,5/12-G-3,5

Plug Connector: Phoenix MC 1,5/12-ST-3,5

Recommended Wire Gauge: 0.6mm^2 (20AWG)

Signal Layout:

       Pin 1: GND (Bottom side of drive)

       Pin 2: Analog Command Reference ±10VDC

       Pin 3: DIR-, B-, CCW- Pulse Reference

       Pin 4: DIR+, B+, CCW+ Pulse Reference

       Pin 5: STEP-, A-, CW- Pulse Reference

       Pin 6: STEP+, A+, CW+ Pulse Reference

       Pin 7: Signal Common for Pin. 8, 9, 10, 11.

       Pin 8: Alarm Output

       Pin 9: OnPosition Output

       Pin 10: Absolute Zero Position Index Output

       Pin 11: Drive Disable Input

       Pin 12: Drive Internal +5VDC Supply (Top side of drive nearest to JP5)

▌ JP4 Encoder Feedback Port

Connector Type: 2.54mm Pitch Rectangular

Drive Header: Molex 70553-0038

Plug Connector: Molex 50-57-9404

Recommended Wire Gauge: 0.3mm^2 (AWG22)

Signal Layout:

        Pin 1: +5VDC Supply

        Pin 2: S+

        Pin 3: S-

        Pin 4: Gnd

▌ JP5 Servo Motor Power

Connector Type: 5.00mm Pitch Terminal Block

Drive Header: Phoenix MSTBA 2,5/ 3-G

Plug Connector: Phoenix MSTB 2,5/ 3-ST

Recommended Wire Gauge: 0.8mm^2 (AWG18)

Signal Layout:

        Pin 1: A Phase

        Pin 2: B Phase

        Pin 3: C Phase

Motor frame should be grounded through Drive Body Grounding Terminal [M3] located on heat sink side. Crimp an M3 terminal lug onto the servo motor frame wire (Yellow/Green) and attach the lug to the drive body grounding terminal. Do not use a screw longer than 5mm.

2.3 JP3 Main I/O Details

> ## ⚠ WARNING
>
> ● Note the directionality of the JP3 connector and pins before making connections. Pin1 is located nearest to the bottom of the servo drive. Pin12 is located nearest terminal JP5 (Servo Motor Power).

■ Terminal Layout

DYN2 AC Servo Drive

DYN2MS-02F-0116A17
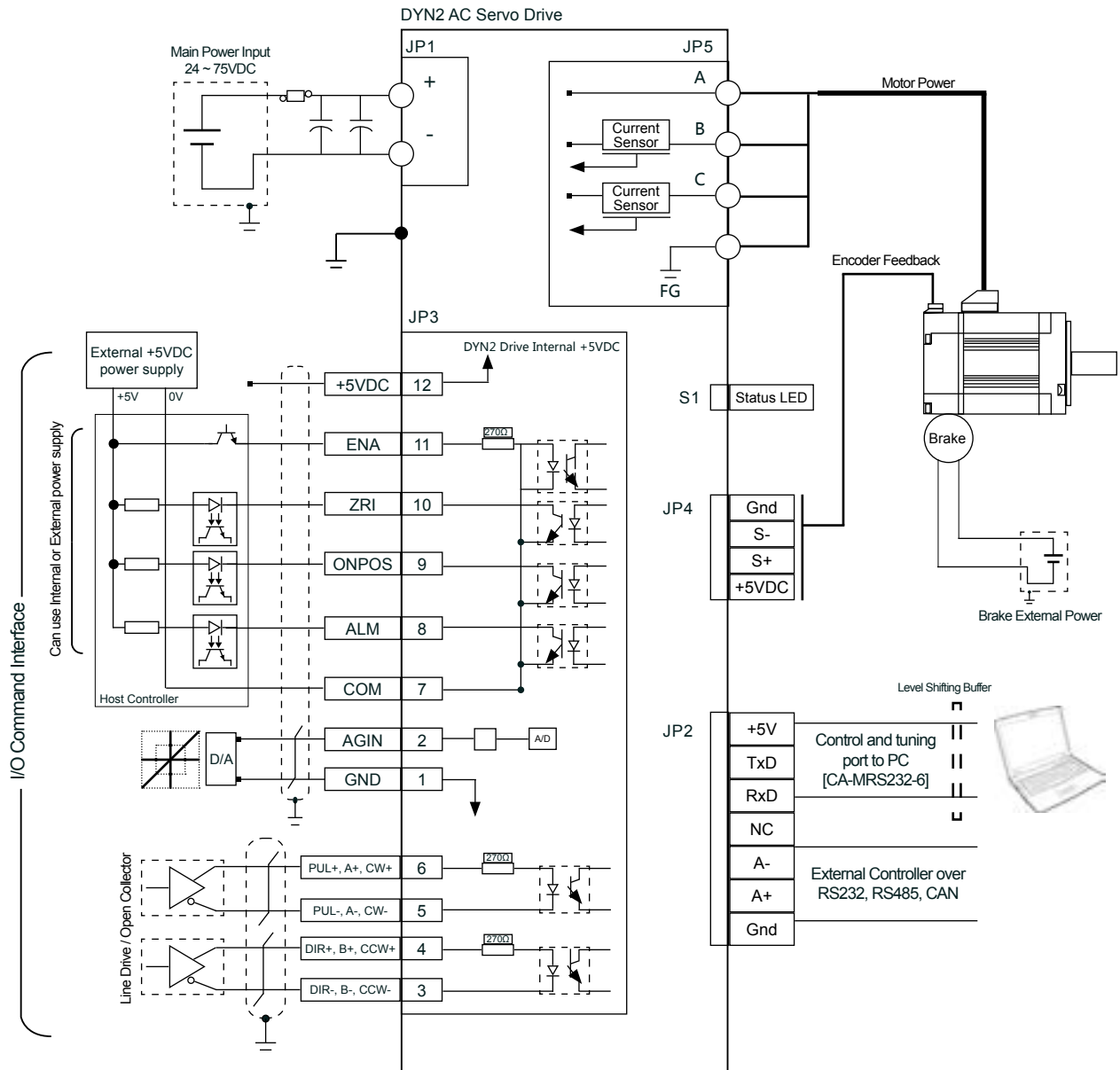
■ JP3 Signal Specification

Refer to Section 2.4 JP3 I/O Connection Circuit for example connection diagram. Standard I/O levels are +5VDC±%10. Contact DMM if the controller uses 12~24VDC level I/O.

Pin No.	Signal	Symbol	Type
12	Drive Internal +5VDC Supply	+5VDC	Output

Description	Connection Circuit
- Drive internal +5VDC output - Max Current Draw: 50mA - Relative ground side with JP3 Pin.1	N/A

Pin No.	Signal	Symbol	Type
11	Drive Disable Input	ENA	Input

Description	Connection Circuit
- Apply +5VDC between Pin.7 Common to Disable servo drive - Motor coasts when disabled (shaft free) - Disable clears all pulse/analog commands - Disable clears all position error - Max. Voltage: +5VDC±%10 - Max. Current: 20mA	[A] *See section 2.4

Pin No.	Signal	Symbol	Type
10	Absolute Zero Position Index Output	ZRI	Output

Description	Connection Circuit
- Transistor ON (Signal LOW) if servo on Zero Position. - Triggered at signal falling edge - Zero Position output fixed to one mechanical motor shaft position per revolution. Accuracy maintained by absolute encoder. - Used for precision zeroing, or indexing applications. - Max. Voltage: 30V - Max. Current: 30mA	[B] *See section 2.4

DYN2MS-02F-0116A17

Refer to Section 2.4 JP3 I/O Connection Circuit for example connection diagram.

Pin No.	Signal		Symbol	Type
9	OnPosition Output		ONPOS	Output

Description	Connection Circuit
- Transistor ON (Signal LOW) if servo Off Position. - Transistor OFF (Signal HIGH) if servo On Position. - Servo On Position if motor position error within value set by *OnPosRange* parameter. - Max. Voltage: 30V - Max. Current: 30mA	[B] *See section 2.4

Pin No.	Signal		Symbol	Type
8	Servo Alarm		ALM	Output

Description	Connection Circuit
- Transistor ON (Signal LOW) if servo drive alarmed or faulted - Servo drive triggers protective alarm relative to Current, Voltage, Temperature, Over Power, Position Lost Follow - Max. Voltage: 30V - Max. Current: 30mA	[B] *See section 2.4

Pin No.	Signal		Symbol	Type
7	Common		COM	Output

Description	Connection Circuit
- JP3 I/O Pin. 8, 9, 10, 11 Control Signal Common.	[A] [B] *See section 2.4

Refer to Section 2.4 JP3 I/O Connection Circuit for example connection diagram.

Pin No.	Signal	Symbol	Type
6	STEP+, A+, CW+ Pulse Reference	STEP+	Input
5	STEP-, A-. CW- Pulse Reference	STEP-	
4	DIR+, B+. CCW+ Pulse Reference	DIR+	
3	DIR-, B-. CCW- Pulse Reference	DIR-	

Description	Connection Circuit

- Position command reference pulse input
- Compatible pulse form include:
    ♦ Pulse + Direction
    ♦ A/B phase quadrature with 90° phase differential
    ♦ CW + CCW
- Max. input pulse frequency: 500kHz
- Max. Voltage: +5VDC±%10
- Max. Current: 20mA
- Line Drive / Open Collector circuit on Controller Side
- Input pulse electronically scalable with *GEAR_NUM* parameter
	[C]
	*See section 2.4

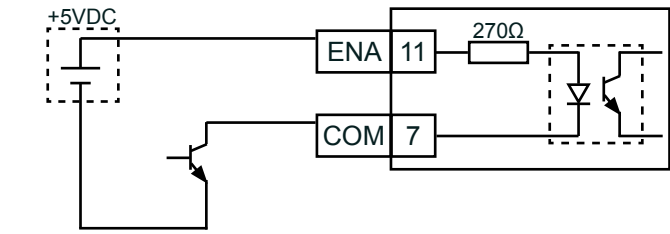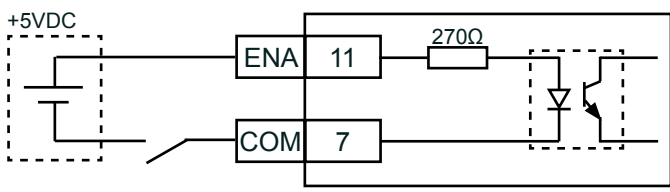| Pin No.	Signal	Symbol	Type
2	Analog Command Reference	AGIN	Input
1	Ground	GND	N/A

Description	Connection Circuit
- Analog command reference for Speed/Torque servo mode - Voltage reference ±10VDC - ±12VDC max input voltage - Max current: 0.6mA	[D] *See section 2.4

DYN2MS-02F-0116A17

■ Type [A] Connection Circuit - General Input Circuit

Applicable Signals:

Pin No.	Signal	Symbol	Type
11	Drive Disable Input	ENA	Input

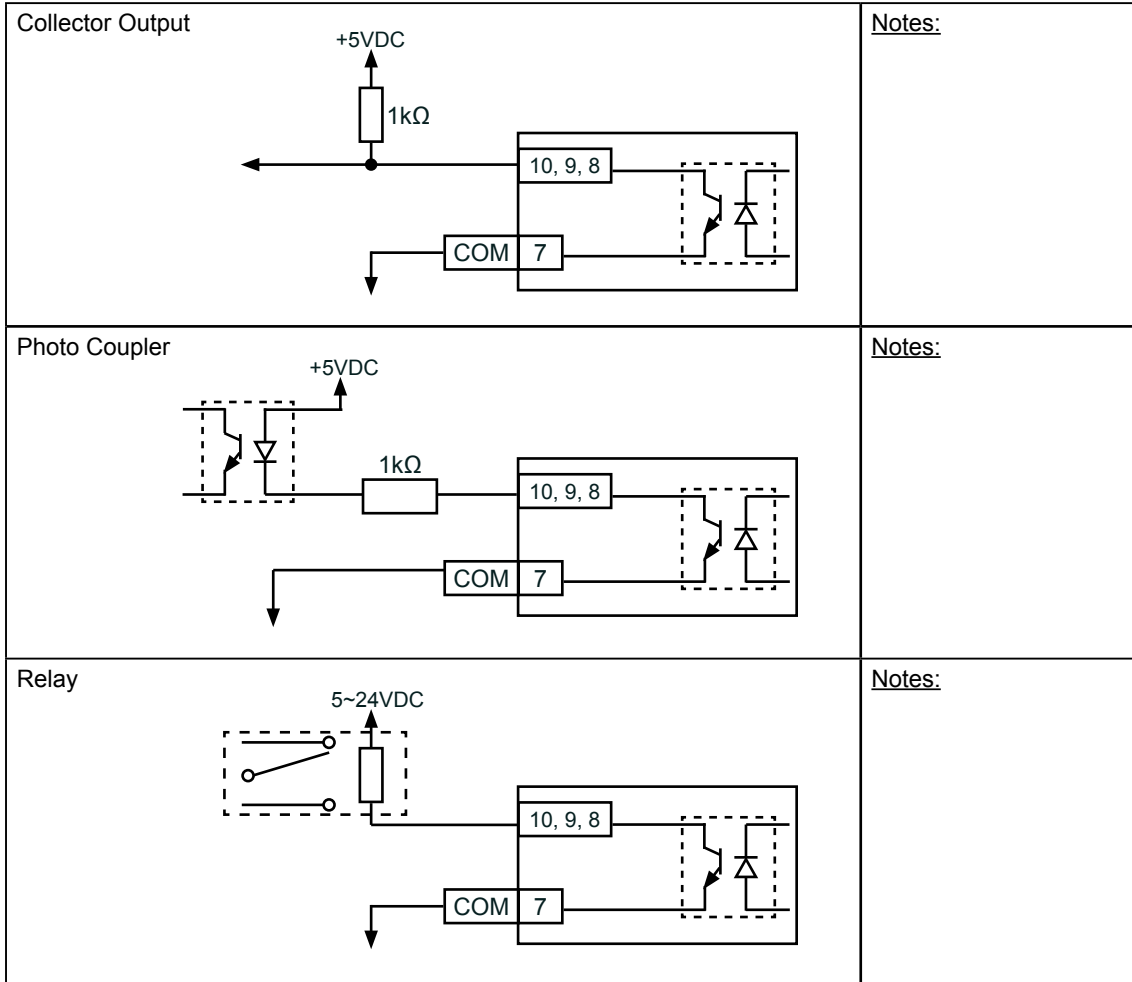| Open Collector	Notes:
	- Sink circuit shown. Source circuit can also be used.
Relay/Switch	Notes:

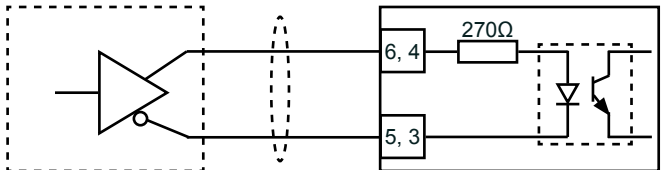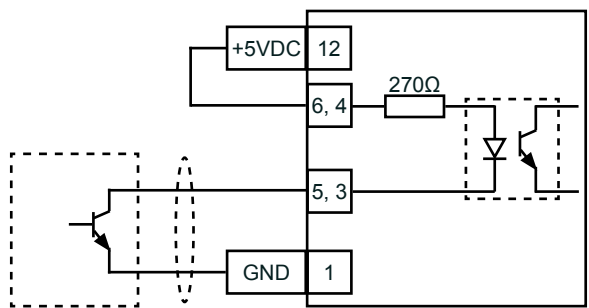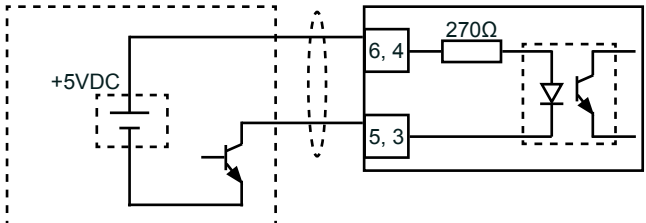■ Type [B] Connection Circuit - General Output Circuit

Applicable Signals:

Pin No.	Signal	Symbol	Type
10	Absolute Zero Position Index Output	ZRI	Output
9	OnPosition Output	ONPOS	Output
8	Servo Alarm	ALM	Output
7	Common	COM	N/A

	Notes:
Collector Output +5VDC 1kΩ 10, 9, 8 COM 7	Notes:
Photo Coupler +5VDC 1kΩ 10, 9, 8 COM 7	Notes:
Relay 5~24VDC 10, 9, 8 COM 7	Notes:

■ Type [C] Connection Circuit - Position Reference Pulse Input

Applicable Signals:

Pin No.	Signal	Symbol	Type
6	STEP+, A+, CW+ Pulse Reference	STEP+	Input
5	STEP-, A-. CW- Pulse Reference	STEP-	
4	DIR+, B+. CCW+ Pulse Reference	DIR+	
3	DIR-, B-. CCW- Pulse Reference	DIR-	

Line Driver	Notes: - Twisted pair cable with shield grounded on receiver side.
Open Collector (DYN2 Internal Power Supply)	Notes: - Twisted pair cable with shield grounded on receiver side.
Open Collector (External Power Supply)	Notes: - Power supply provided by host controller or external source. - Twisted pair cable with shield grounded on receiver side.

■ Type [D] Connection Circuit - Analog Command Reference Input

Applicable Signals:

Pin No.	Signal	Symbol	Type
2 1	Analog Command Reference Ground	AGIN GND	Input N/A

Notes:
- Twisted pair cable with shield grounded on receiver side.

The DYN2 servo drive has a minimum operation input of +24VDC and max input of +75VDC. The servo drives internal over-voltage alarm is triggered at +80VDC input and will shut down at this level. Consider the voltage/speed gradient of the servo motor when selecting power supplies.

A smoothing (reservoir) capacitor is recommended after the DC power supply. The recommended capacity is 100V 1,000uF per kW of motor load. Connect a fuse before the servo drive according to the circuit size.

| Servo Drive Model	Max. Motor Capacity	Recommended Fuse
DYN2-T1A6S-00	200W	15A
DYN2-TLA6S-00	750W	30A

⚠ WARNING

● DO NOT reverse the polarity of the DC input power. Reversing the polarity will permanently damage the servo drive and may cause electric shock. Ensure polarity is correct before powering ON the servo drive.
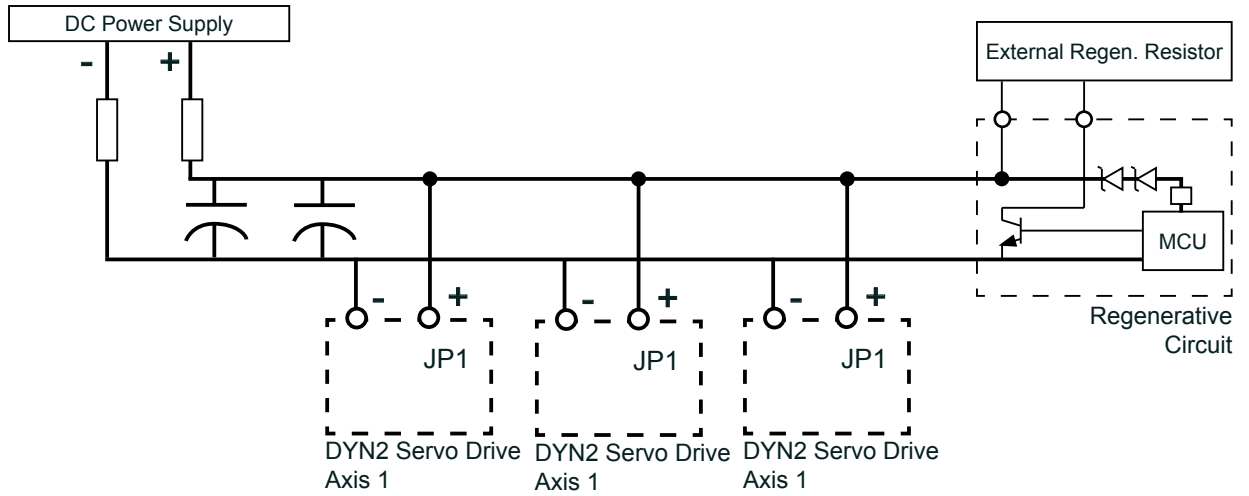
■ Single Axis

■ Multi-Axis (Common DC Bus)
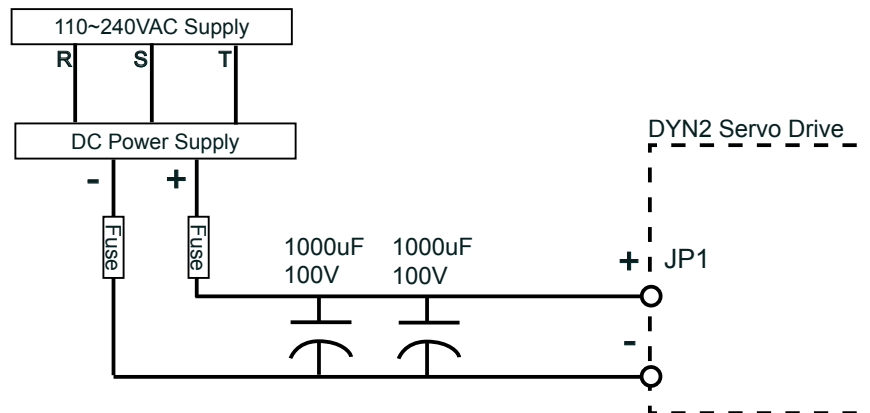
[*1] Select fuse and smoothing capacitor size according to number of drives on DC bus.
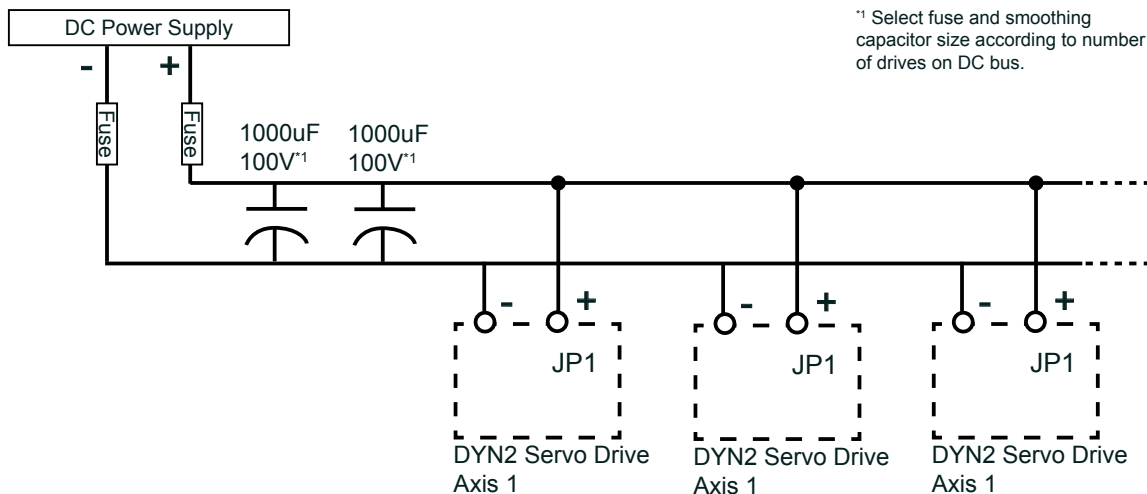
■ Regenerative Circuit

An external regenerative circuit may be needed for applications with high load inertia deceleration. Contact DMM for DYN2 regenerative circuit requirements.

3   START UP

3.1 Mounting and Installation

The DYN2 servo drive can be mounted vertically or horizontally (vehicle mount). The servo drives should be mounted by its rear chassis to an electrically conductive metal panel or plate. When mounting multiple servo drives, at least 1mm clearance should be left between each unit. The small size of the DYN2 servo drive is compatible with modular mounting. It can be placed adjacent to other devices with 1mm clearance. Also consider the size of the connectors and cables in front of the servo drive when mounting.

The control cabinet internal temperature should not exceed 40°C. If using a fan to cool the servo drives, the air flow should parallel the direction of the heat sink fin. The servo drive internally do not have a cooling fan. Contact DMM if servo drives need to be placed adjacent without spacing.

DYN2MS-02F-0116A17
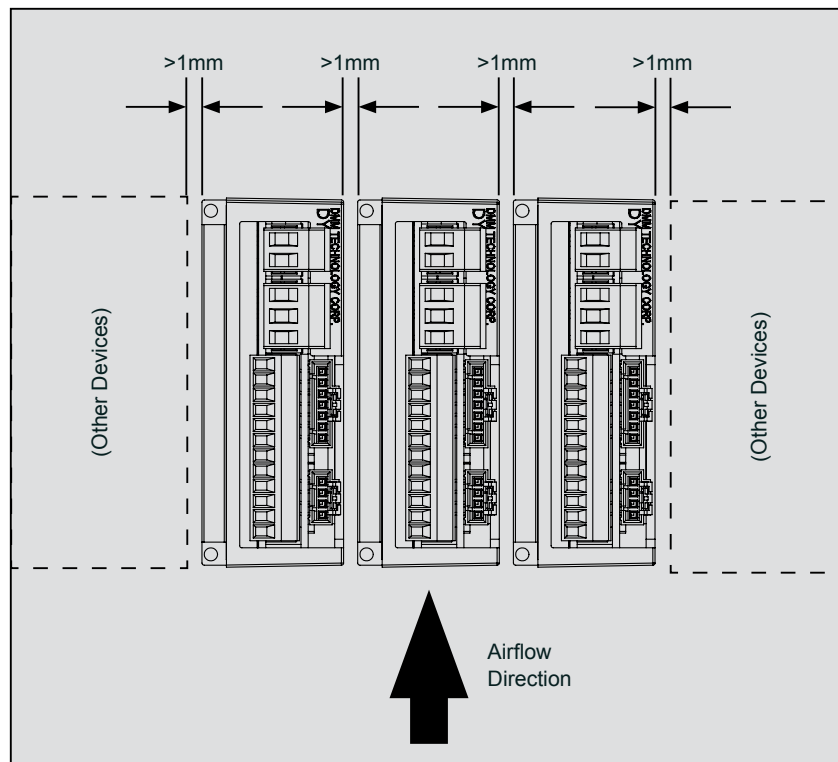
■ Power ON Timing

After servo drive power ON, make sure there is at least 150ms time before sending pulse or analog command to servo drive.
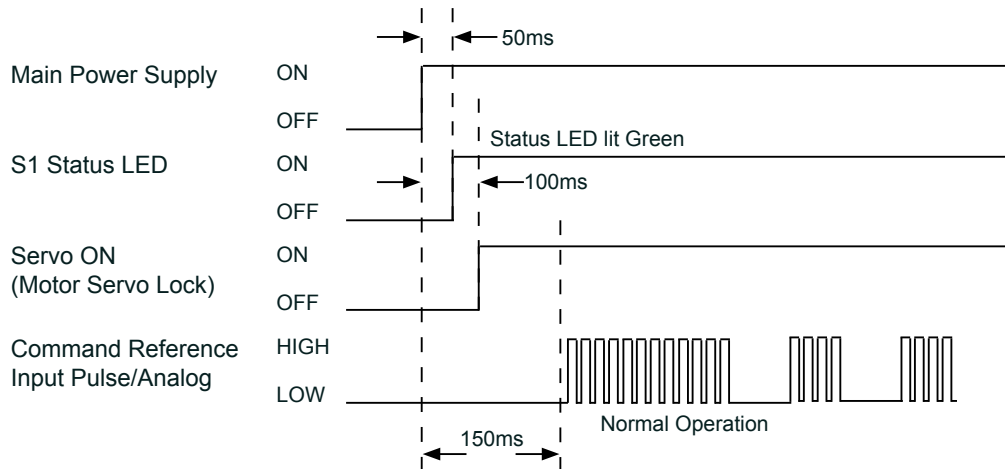
◆ Main Power Supply Cycle

Do not cycle the main power supply quickly as internal power electronics may be permanently damaged. The main power should be turned on once during each operation cycle and should not be controllable by software.

◆ Power Off Residual Voltage

After drive power off, the user should wait 60 seconds before touching the servo drive. A residual voltage may remain in the servo drive after immediate power off and 60 seconds is needed for full discharge. This time may be longer if a larger smoothing capacitor is connected to the input power line.

The residual voltage may cause the servo motor to rotate for a short period (<1 second) after immediate power off. Consider this effect for emergency situations and take safety precaution to prevent damage to personnel, equipment or machine.

DYN2MS-02F-0116A17

■ Servo Disable / Enable Timing

When using the ENA signal to disable the servo drive to coast the servo motor, do not cycle this input rapidly ON/OFF. If the signal is cycled too fast, the servo drive will not have enough time to initialize the control program during Enable and can cause unwanted or dangerous results. Ensure that in the control program, the below timing is satisfied. Once Disabled, do not Enable the servo drive during motor coast or any time motor shaft is rotating, make sure motor shaft is completely stopped before Enable.
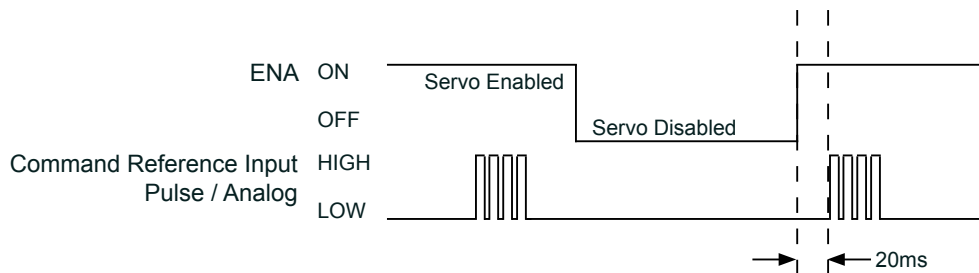
3.3 DMMDRV Software Communication

■ Version 1.0

♦ PC Running Requirements
   Win98/XP/2000/Vista/7
   250Mhz CPU
   64MB RAM
   250MB Hard Disk Space

The servo drive should be powered up with the servo motor encoder feedback and motor power cables connected. The servo motor shaft will be servo-locked when powered ON. Connect the RS232 tuning cable from port JP2 to host PC.

♦ DMMDRV Start Up
   1) Open the DMMDRV.exe executable
   2) Select "COMSET" --> "COM PORT"
   3) Change the port number to the servo drive connected RS232 port, then select "OK"
   4) Select "SERVO SETTING"
   5) Select DYN2 -DRIVER
   6) Press Read on the Setting driver parameters and mode dialogue box. After approximately 1~2 seconds, the on-screen parameters will change according to the current internal parameter settings of the connected servo drive. Ensure that the Driver Status indicates ServoOnPos to indicate that the drive has closed the position loop with the motor and is fully operational.

⚠ WARNING

 ● During Test movement procedures, the servo motor can rotate very quickly in either direc-
  tion. Ensure that the servo motor is free to rotate and no objects are attached to or is near
  the motor shaft. Secure the motor by its flange.

♦ Test Movements
   1) Select "RS232" under the command input mode option, then click "SAVE ALL" to save this setting.
   2) Under the Test Motions menu, the user can select one of 4 test motions to JOG, STEP, SINE or TRIM the servo motor. Only one test motion profile can be run at a time, use the radio buttons below each section to select the movement profile.

■ Version 328.1

♦ PC Requirements

       Operating System: Windows XP SP3 or higher

       *Recommended: Windows 7 (32-bit / 64-bit)

       Processor: Pentium 1 GHz or higher

       RAM: 512 MB or more

       Framework: .NET Framework 4 or higher

       Minimum disk space: 60MB

*See User Manual DSFEN for complete instructions:
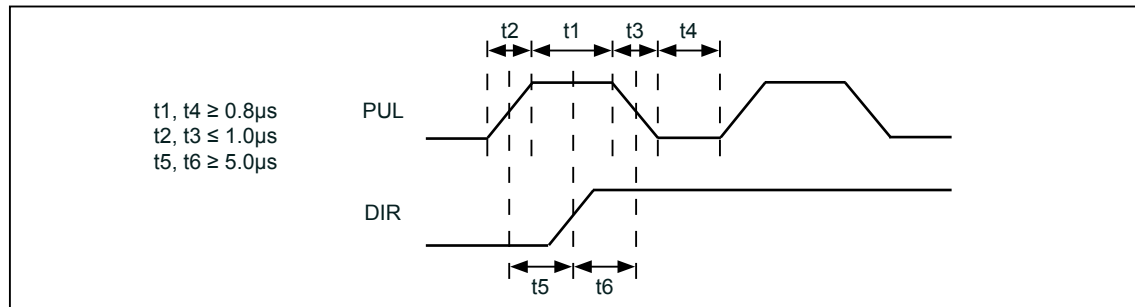
4   OPERATION

4.1 Position Servo Mode

◼   Pulse Specifications

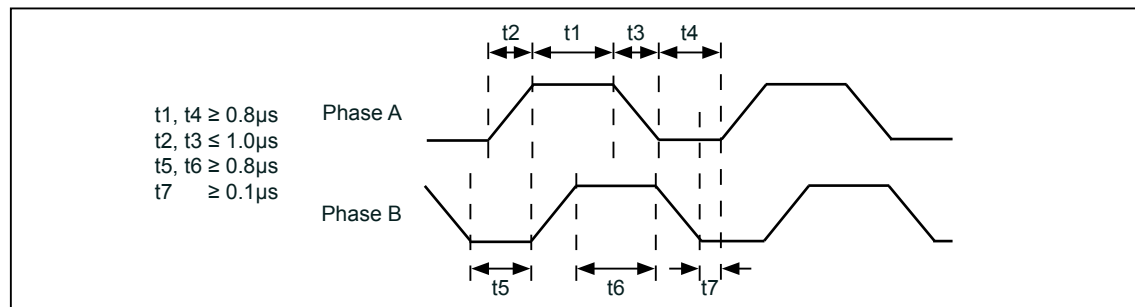Voltage: +5VDC ± %10 (Contact DMM if higher level such as 12/24VDC is required)
Max pulse frequency: 500kHz
Minimum pulse width: 0.8µs

♦ Pulse + Direction

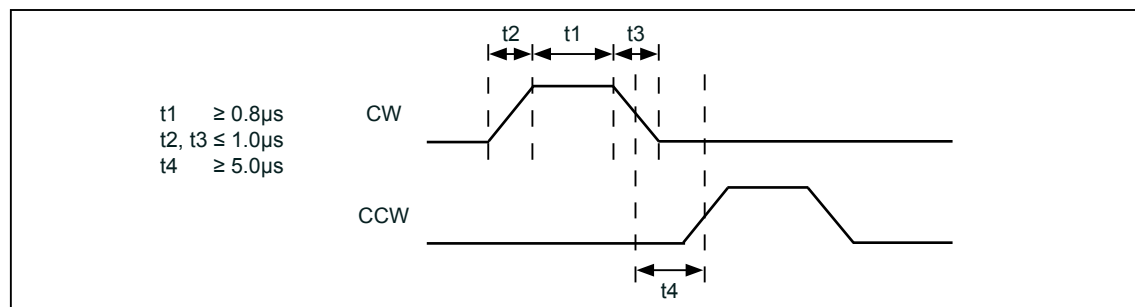t1, t4 ≥ 0.8µs
t2, t3 ≤ 1.0µs
t5, t6 ≥ 5.0µs

PUL

DIR

♦ A/B phase quadrature with 90° phase differential

t1, t4 ≥ 0.8µs
t2, t3 ≤ 1.0µs
t5, t6 ≥ 0.8µs
t7   ≥ 0.1µs

Phase A

Phase B

♦ CW + CCW

t1   ≥ 0.8µs
t2, t3 ≤ 1.0µs
t4   ≥ 5.0µs
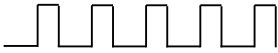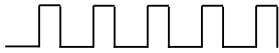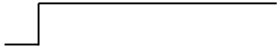
CW

CCW
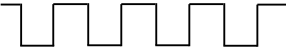
■   Reference Pulse Format

The DYN2 servo drive accepts FORWARD reference as CLOCKWISE motor shaft rotation as viewed from motor shaft side.
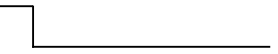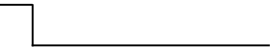
♦ Pulse + Direction

Forward Reference	Reverse Reference
PUL+ JP3-6	PUL+ JP3-6
DIR+ JP3-4	DIR+ JP3-4

♦ A/B phase quadrature with 90° phase differential

Forward Reference	Reverse Reference
A+ JP3-6	A+ JP3-6
B+ JP3-4	B+ JP3-4
A Leads B	B Leads A

♦ CW + CCW

Forward Reference	Reverse Reference
CW+ JP3-6	CW+ JP3-6
CCW+ JP3-4	CCW+ JP3-4

■   Connection Example

♦ Line Drive Output
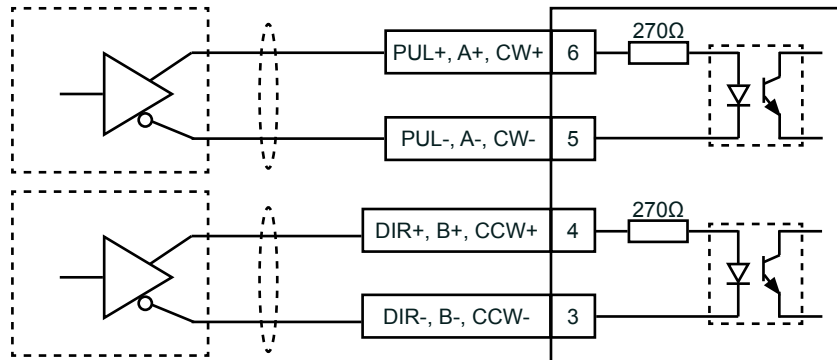
♦ Open Collector Output - Internal Power Supply
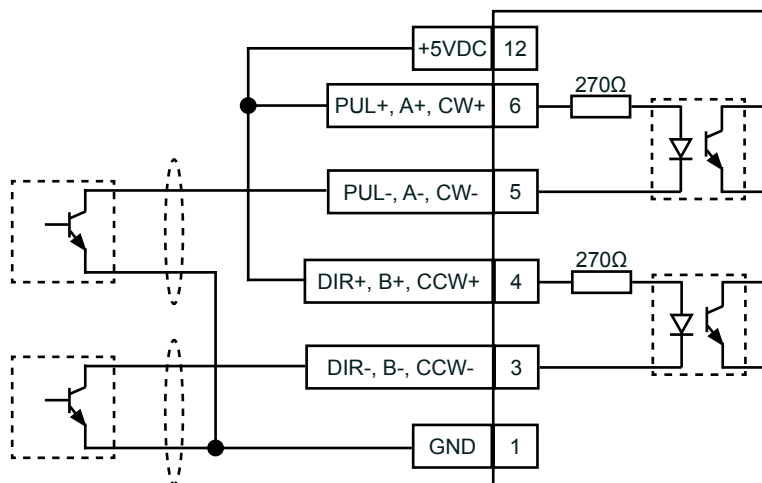
♦ Open Collector Output - External Power Supply

DYN2MS-02F-0116A17
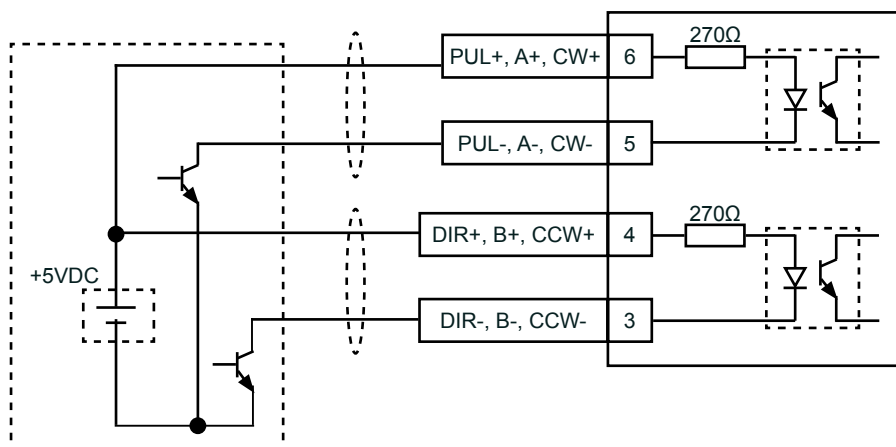
■　　　Electronic Gearing (*GEAR_NUM* Parameter)

Gear number is set from 500 to 16,384, default value is 4,096. Gear number provides an electrical gear ratio: 4096 / Gear_Num, from 0.25 ~ 8.192. For example, if Gear number = 4,096, the 16,384 input counts from pulse will turn motor exactly one revolution. If Gear number = 500, 2,000 pulses will turn motor one revolution.

For analog input in position servo mode, the analog input is from 0~10VDC range, by using the Gear Number, 0~10VDC analog input can turn motor from 0~90*4,096/Gear number (degrees). The gear number has the same effect on the serial Point to Point movement or RS232 command input mode. Gear number parameter is only effective for position servo mode.

■　　　Servo In Position Output Specifications (*ONPOS*)

 On position range is a value used for determining whether the motor have reached the commanded position or not. That on position range is selectable according to customer's requirement. Suppose the Pset is the commanded position, and Pmotor is the real motor position, if

$$|Pset - Pmotor| <= OnRange$$

it is said motor is ON the commanded position, otherwise not. That OnRange is set from 1~127. The real position on range is: OnRange * 360(deg)/16,384. Set mouse curser into the OnPosition edit box, input the desired on position value, then click the save button, On position value will be sent to the Drive with all other parameters. The ONPOS output (JP3-9) will be HIGH if motor in position and LOW if motor off position.

■　　　Servo Position Error Accumulation

The servo drive's internal position error decides the status of the On Position signal and the Lost Phase servo drive alarm.

The On Position signal is triggered (LOW) when the servo position error is within the OnPosRange set in the DMMDRV program. The Lost Phase alarm is triggered when the servo motor is physically 90° or more out of position for ~2 seconds.

The servo position error is cleared when the drive is disabled using the ENA input and does not accumulate when the drive is disabled.

4.2 Speed Servo Mode

In speed servo mode, the DYN2 servo drive takes command from an external ±10VDC analog reference voltage from the host controller to drive a linear proportional motor speed.

In speed servo mode, the torque output depends on the load on the servo motor and determined by the motor feedback. Design the system so it can withstand the peak torque of the motor in use.

■   Control Reference

The DYN2 servo drive accepts FORWARD reference as CLOCKWISE motor shaft rotation as viewed from motor shaft side. Positive reference voltage rotates the servo motor in the FORWARD (CLOCKWISE) direction and negative reference voltage rotates the servo motor in the REVERSE (COUNTER CLOCK-WISE) direction.
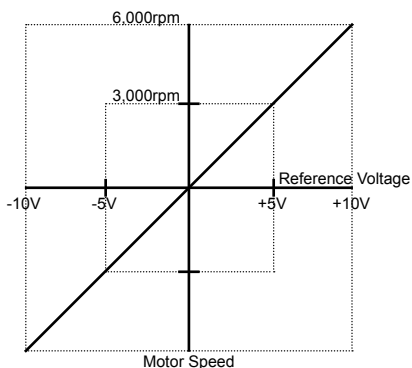
Reference Voltage	Motor Speed	Reference Direction	Motor Direction
+10V	6,000rpm	FWD	CW
+5V	3,000rpm	FWD	CW
-3V	1,800rpm	REV	CCW

DYN2MS-02F-0116A17

■   Acceleration / Deceleration Soft Start

In Speed Servo Mode, the *Max Acceleration* parameter in the servo drive can be used to soft start/stop the servo motor. Since the speed command is sent as a rough step reference, it is often desirable to smooth out the servo motor's movement dynamics. Without soft start, the servo motor can accelerate/decelerate too instantaneously. Soft start creates a smooth s-curve motion.

The relation to physical acceleration / deceleration time is measured as the rise time from 10% of the target speed to 90% of the target speed.

Rise from 10% to 90% time = $59.98/(Max\ Acceleration)^2$ seconds
Physical acceleration time = $1.2 * 59.98/(Max\ Acceleration)^2$ seconds

Rough Motion   Smooth Motion!

Max Acceleration
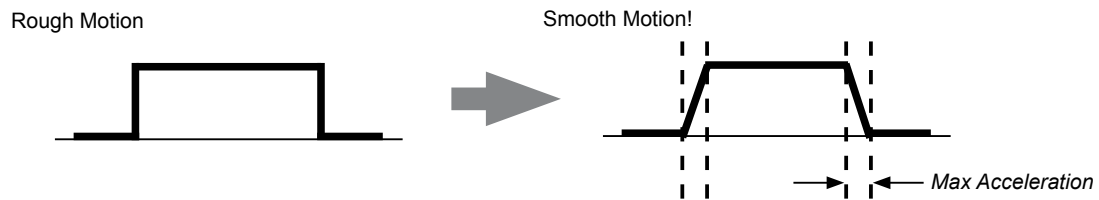
■   Torque Filter Constant

TrqCons is a first order low-pass filter used to smooth torque delivery in speed servo mode which improves stability and accuracy of servo motor speed. The bigger value means wider frequency range of that filter. That filter can be expressed as:

$a / (S + a)$, here $a = 26*TrqCons$; if TrqCons = 100, then a = 2600.

The filter is used to make the torque sent to the servo torque loop more smooth especially for the heavier load when bigger SpeedGain setting is used. If a very quick response servo with small load is desirable, the bigger value or even the value 127 should be used to ensure stability and fast dynamic follow.

The Torque Filter Constant parameter should only be used in speed servo mode. Leave this parameter at "127" in position servo mode.

In torque servo mode, the DYN2 servo drive takes command from an external ±10VDC analog reference voltage from the host controller to drive a linear proportional output current.

▉ Control Reference - [1] Capacity Model: DYN2-T**1**

The DYN2 servo drive accepts FORWARD reference as CLOCKWISE motor shaft rotation as viewed from motor shaft side. Positive reference voltage rotates the servo motor in the FORWARD (CLOCKWISE) direction and negative reference voltage rotates the servo motor in the REVERSE (COUNTER CLOCK-WISE) direction.
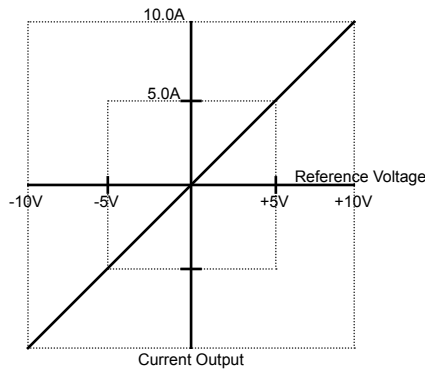
Reference Voltage	Output Current	Reference Direction	Motor Direction
+10V	10.0A	FWD	CW
+5V	5.0A	FWD	CW
-3V	3.0A	REV	CCW

▉ Control Reference - [L] Capacity Model: DYN2-T**L**

The DYN2 servo drive accepts FORWARD reference as CLOCKWISE motor shaft rotation as viewed from motor shaft side. Positive reference voltage rotates the servo motor in the FORWARD (CLOCKWISE) direction and negative reference voltage rotates the servo motor in the REVERSE (COUNTER CLOCK-WISE) direction.
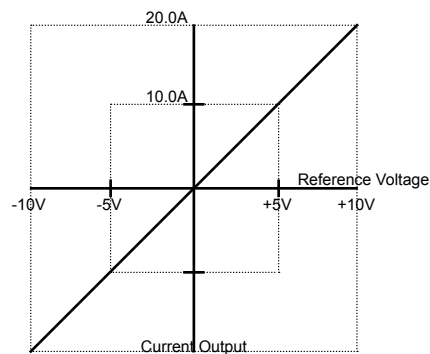
Reference Voltage	Output Current	Reference Direction	Motor Direction
+10V	20.0A	FWD	CW
+5V	10.0A	FWD	CW
-3V	6.0A	REV	CCW

DYN2MS-02F-0116A17

4.4 RS232 Command Input Mode

The RS232 port is always active after power on for DYN-series servo drive, that active RS232 port could be used for reading and setting Drive parameters and status, also could be used for sending point to point position command if the RS232 mode is selected for position command input.

If the position command input mode is selected as Pulse mode or Analog mode, the RS232 port is still active as mentioned above but it only can be used for reading and setting Drive parameters. The RS232 port could be easily accessed by using the GUI interface DMMDRV.exe after the connection between PC and the Drive's RS232 port. This is the easiest way to tune up the servo and make some test movements. The RS232 port could be accessed by other microcontroller, or DSP if sending and reading data by using DYN Drive's RS232 protocol.

The PC or DSP is working as Master and the servo drive is always as slave. Several servo drives could be linked for a serial network integrated multi-axis control.

See (Appendix A) for DYN2 servo drive RS232 protocol definitions.

4.5 Absolute Zero Position Index Output (ZRI)

The ZRI signal is output once per motor revolution to facilitate servo homing and indexing functions. ZRI pulse can also be used to count motor revolutions or monitor servo motor speed. Accuracy of each pulse is maintained by 14/16-bit absolute encoder. The mechanical output position of ZRI may vary between each servo motor. It can also be used to compensate for mechanical or ball screw backlash. A calibration procedure is necessary to set the absolute ZRI position in the controller.

The user should calibrate the position of the ZRI output with respect to the target mechanical position. The falling edge of the ZRI output (JP3 Pin.10) should be used as the trigger. Pulse width and rising edge of ZRI should not be used as trigger.
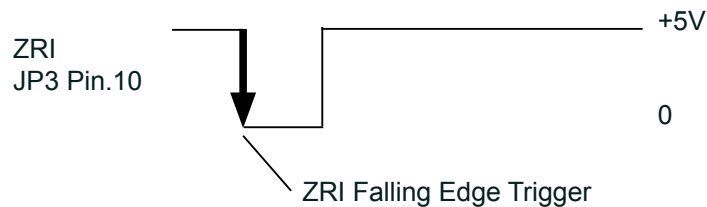
5   PARAMETERS AND TUNING

5.1 Parameters Outline

The following parameters are adjustable by connection through RS232 or USB interface from the servo drive to the PC. No matter the command mode, the JP2 RS232 port is always active for parameter setting and drive configuration.

The Drive configuration and servo cons are stored in the EEPROM of servo drive when the save button is pushed or parameters setting is issued through the serial communication.

The guaranteed write cycle for the EEPROM is 1 million times. Do not use serial communication to constantly change the drive parameters as this will decrease servo drive life span. Major parameter change and setting should only be done during initial testing and tuning. Actual drive operation should not require constant parameter changes unless changing servo control modes on the fly through RS232.

Parameter Name	Setting Range	Details	Applicable Servo Mode
Main Gain	[1 : 127]	The main gain for the servo loop, usually to be increased as the motor load increases. The bigger value of MainGain means wider frequency range of servo loop relatively.	Position Speed Torque RS232
Speed Gain	[1 : 127]	The speed gain for the servo loop, usually to be increased as the motor load increases. The bigger value of speed Gain means narrower frequency range of servo loop relatively. Physically, heavier loads or higher inertia loads should have lower dynamic ability, so the servo loop frequency range should be more narrow by using bigger value of Speed Gain. If the Speed Gain is too high, there will be some loud noise because the torque command will be too coarse, not smooth, the smaller Torque Constant (see TrqCons) could be used to attenuate this noise.	Position Speed Torque RS232
Integration Gain	[1 : 127]	There is an integrator in the servo loop to ensure the error between position command and real position be zero during the steady state. Also that integrator will let servo have more ability to attenuate the outside disturbance torque. The bigger value of IntGain, the more ability of the servo to attenuate the outside disturbance torque. Integration Gain should be decreased for heavier loads or higher inertia loads.	Position Speed Torque RS232
Torque Constant	[1 : 127]	TrqCons is a first order filter constant, the bigger value means wider frequency range of that filter. That filter can be expressed as : a / (S + a), here a = 26*TrqCons, if TrqCons = 100, then a = 2600. That filter is used to make the torque sent to torque loop more smooth, especially for heavier loads when bigger SpeedGain is used. If a very quick response servo with small load is desirable, a bigger value or even the value 127 should be used to ensure the stability and dynamic performance.	Speed Torque RS232

DYN2MS-02F-0116A17

Parameter Name	Setting Range	Details	Applicable Servo Mode
Max Acceleration	[1 : 127]	Determine the S-curve acceleration when using RS232 mode to make point to point motion linear/circular. Also controls the response time of the first order low pass filter in speed and torque servo control (soft start).	RS232 Speed Torque
Max Speed	[1 : 127]	Determine the S-curve max speed when using RS232 mode to make point to point motion linear/circular.	RS232
Driver ID	[1 : 126]	Every drive has a unique ID number, which can be assigned or read out by using ServoSetting dialog box. Applicable when RS485net box not checked and there is only one Drive connected through the RS232 port. The default ID number for every Drive is 0. That ID number can be used for the network connection of RS485 or for drive unit identification purposes. When RS485net box is checked and there are more than one Drive connected to the RS485/232 network, only the setting for the Drive with the indicated ID number in the ServoSetting dialog box can be read out or saved.	Position Speed Torque RS232/485
On Position Range	[1 : 127]	On position range is a value used for determining whether the motor have reached the command position or not. That on position range is selectable according to user's requirement. Suppose the Pset is the commanded position, and Pmotor is the real motor position, if $$|Pset - Pmotor| <= OnRange$$ it is said motor is on the commanded position, otherwise not.	Position Speed Torque RS232
Gear Num	[500 : 16,384]	The amount of motor travel with reference to the number of input pulses is set using the parameter Gear_Num. The number of reference pulse needed for one complete motor revolution is calculated as, $$One\ motor\ revolution = 4 \times GEAR\_NUM$$ For example, if Gear_Num is set to 4096, then 16,384 pulses are needed from the host controller for the motor to make one complete revolution.	Position RS232

DYN2MS-02F-0116A17

The DYN2 servo drive features simple 3 parameter Gain tuning to achieve optimized smooth performance. The user should adjust the servo gain parameters Main Gain, Speed Gain and Integration Gain until they achieve target response qualities. These parameters are all adjustable using the DMMDRV program.

The built in Adaptive Tuning algorithm optimize servo region of stability relative to load inertia. As long as the 3 gain parameters are close to ideal settings, the servo will always achieve best response.

The overall method of Gain tuning follows as load mass or load inertia increase, the Main Gain and Speed Gain parameters should be increased. If these are set too high, the servo may be over-tuned and start vibrating or become noisy. The parameters should be increased/decreased until the motor smoothly follows command without vibration, noise or oscillations. The user can then fine tune the parameters to make the motor "harder" (faster response, more rigid motion) or "softer" (slower response, smoother motion).

The servo motor should be coupled to the final machine before tuning. Make sure during tuning, the motor is running the load and speed of the final machine or design. The user should use a trial and error method to achieve the desired servo response.

In Speed and Torque servo mode, the Torque Filter Constant parameter can be adjusted to further smooth the torque delivery and improve motor speed accuracy.

♦ Gain Tuning Procedure Flow
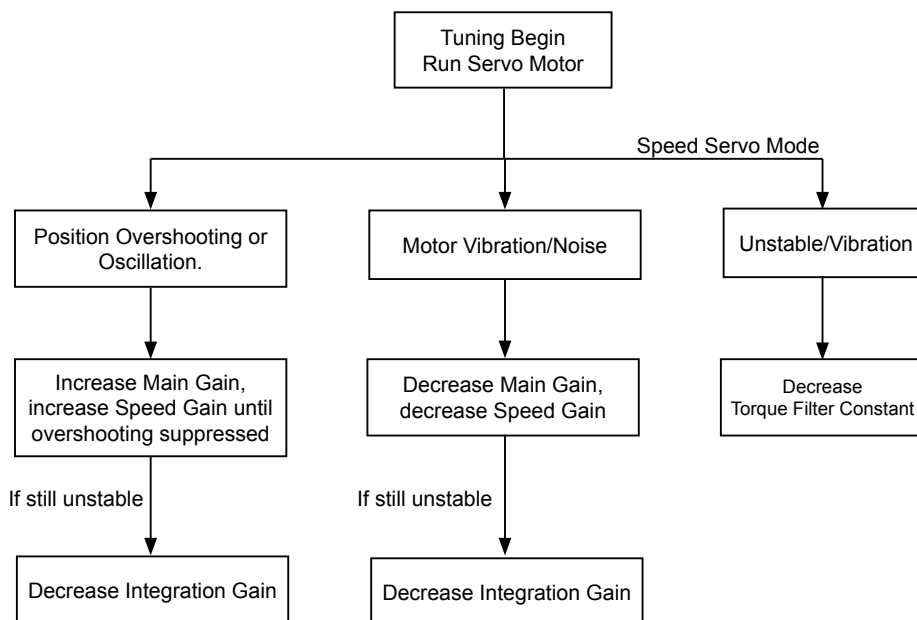
■   Sample Load Type Tunings

♦ Ball Screw

Ball screw systems are mechanically very rigid and stiff. If high resolution pitch (e.g. 5mm or 10mm) the default setting could even be used. The servo drive can be easily tuned using Main Gain, Speed Gain, and Integration Gain. Increase Main Gain, Speed Gain and Integration Gain relative to load mass until target response achieved. Decrease Integration Gain if load inertia is big and system oscillates.

♦ Direct Mechanical Drive (Rigid systems, Robots)

Depending on load mass and inertia, increase Main Gain, Speed Gain and Integration Gain until target response achieved. Decrease Integration Gain if load inertia is big and system oscillates. In speed/torque servo mode, if relative load inertia is very high, the high Speed Gain might increase motor noise, then decrease the Torque Filter Constant to attenuate torque loop noise.

♦ Belt Drive / Pulley

Belt drive or pulley systems are low mechanical rigidity with slower response. Main Gain and Speed Gain should be increased with higher load mass and relative load inertia. Integration Gain should be decreased to give the position loop more time ro react to the low rigidity system.

6   MAINTENANCE

6.1 Alarm Specifications

The DYN2 servo drive is protected by 5 alarms. The S1 status indicator LED will flash to indicate when an alarm is triggered. The specific alarm status can be read using the DMMDRV program.

♦ Internal Driver Status Readout

(1)   Connect the PC to the servo drive JP2 port using RS232 cable
(2)   Press Read on the Setting driver parameters and mode main screen.
(3)   The Driver Status box will display the current status of the Servo Drive.

Alarm	Cause	Recommended Correction
Over Voltage	The internal DC bus voltage has exceeded the allowed maximum levels. The input DC voltage is too high.	- Check and confirm the connections to the servo motor. - Check that the servo motor is driving a mass appropriate to its size. - Check for any mechanical irregularities that might be preventing the motors to move freely. - Add an external regenerative resistor
Over Temperature	The servo drives protective thermal resistor has detected an unusually high temperature inside the drive. The control power transistor temperature is too high.	- Check that the drive's ventilation openings and heat sink are not being blocked. - Consult the servo drive's ambient temperature specifications and check if the operation conditions are met.
Lost Phase	The encoder has detected an irresolvable position error in the motor relative to the command signal.	- Check that the encoder feedback cable is securely plugged from the servo motor to the JP3 port of the servo drive. - Check for any mechanical irregularities that might be preventing the motors to move freely.
Over Power	The servo drive has experienced an output power exceeding the rated value relative to the average value.	- Check and confirm the connections to the servo motor. - Check that the servo motor is driving a mass appropriate to its size. - Check for any mechanical irregularities that might be preventing the motors to move freely.
Over Current	The servo motor cannot move to its command position and there is a backlog of current in the servo drive to try to move the servo motor.	- Check that the encoder feedback cable is securely plugged from the servo motor to the JP3 port of the servo drive. - Check for any mechanical irregularities that might be preventing the motors to move freely.

♦ Alarm Motor Stop

The power to the servo motor will be stopped when an alarm is triggered. Internal servo control turns off and servo motor shaft becomes free. Power still remains in the logic circuit for drive diagnostic and drive status reading. All commands including pulse, analog and RS232 will be ignored and will not accumulate the internal position error.

♦ Alarm Reset

Once servo drive triggers an alarm, the user should use the DMMDRV program to read out the alarm condition then inspect the machine, load or operation for cause to the alarm. The problem should be fixed before re-setting the servo drive and running again. The servo drive main power should be cycled to fully re-set and clear the servo alarm status.

⚠ WARNING

● If the servo motor is coupled to a vertical axis that can drop due to gravity when the shaft becomes free, take measure to prevent injury or damage when the drive alarm is triggered. A motor with brake option may be necessary to stop vertical axis, or any axis acted on by an external force, from dropping or crashing.

6.2 - Drive Maintenance

Do not perform maintenance on the servo drive unless instructed to do so by DMM. The servo drive cover or chassis should never be removed as high voltage components can cause electric short, shock or other damage upon contact. Disassembly, repairs or any other physical modification to the servo drive is not permitted unless approved by DMM.

♦ Regular Inspection

Inspect the servo drive regularly for:
● Dirt, dust or oil on the servo drive - make sure the servo drive cooling duct and heat sink are free from debris
● Environment - ambient temperature, humidity and vibration according to servo drive specification
● Loose screws
● Physical damage to servo drive or internal components

The RS232 port is always active after power on for DYN series drive. This active RS232 port could be used for reading and setting Drive parameters and status and also can be used for sending point to point position command if the RS232 mode is selected for position command input.

This DYN232M integrated motion command includes point-to-point S-curve, linear, arc and circular interpolation for up to 3-axis of coordinated motion. These profiles can be easily executed using dedicated function codes. The DYN servo drive features the most advanced built in S-Curve Generator in the industry to realize point to point S-Curve motion. Response is extremely fast and motion filters are built in to optimize stability and provide smoothest motor response. Featuring Dynamic Target Position Update (DTPU) technology, target position can be instantaneously changed (without current command completion) and robot movements can be realized with much faster cycle time and higher universal efficiency.

If the position command is selected as other modes, such as PULSE/DIR, CW/CCW, SPI or Analog mode, the RS232 port is still active as mentioned above but only can be used for reading and setting drive parameters and reading and setting drive status registers (Section 7.3).

The RS232 port can be accessed by a variety of host controllers including PC, microcontroller, FPGA, Arduino or motion controller. The host device is working as a master and the servo drive is always working as a slave. Several drives can be linked for a serial network in RS485.

RS232 Functions Include:
- ♦ Reading and changing servo drive parameters
- ♦ Reading and monitoring servo drive status including alarm, busy, in position, enable etc.
- ♦ Reading and monitoring servo drive configuration including servo mode, incremental/absolute mode, command mode, enable etc.
- ♦ Absolute encoder homing
- ♦ Absolute encoder position monitor: 16-bit absolute, 32-bit multi-turn
- ♦ Initiate generic profiles ConstSpeed, Square Wave, Sine Wave
- ♦ DYN232M motion control commands including:
    - ♦ S-Curve point to point
    - ♦ 3-axis coordinated linear motion
    - ♦ 3-axis coordinated circular motion (arc, circle, oval)
    - ♦ Incremental (relative) or absolute modes

Example Host Controllers:
- Microcontroller/Embedded Controller
- PC (windows serial port via C/C++/C#, VB, Java etc.)
- PLC/HMI with serial output
- Arduino

The sample code in Section *7.7A Appendix : C++ Code for Serial Communication Protocol* should be used extensively to efficiently and accurately generate the RS232 data packet. Each subroutine function automatically generates data packet structure for sending command and reading from DYN servo drive.

Never use serial communication to set the Drive configuration or parameters at a fast rate. This will cause servo drive EEPROM busy in writing parameters all the time and also shorten it's lifetime. The guaranteed parameter writing cycle for EEPROM is 1 million times. Once a group of parameters and configuration are set, use it until next necessary change.

■   Connector Specifications

Connection: JP2
Connector Type: 2.54mm Pitch Rectangular
Drive Header: Molex 70553-0041
Plug Connector: Molex 50-57-9407
Recommended Wire Gauge: 0.3mm² (AWG22)

In order to connect JP2 with a PC's RS232 port, an intermediate level shift buffer is necessary [buffer component: ADM232]. The CA-MRS232-6 and CA-MTUSB-60 tuning cables has the level shift buffer built-in. RxD and TxD RS232 signal from connector JP2 is TTL/CMOS level.

Do not connect servo drive directly to PC RS232 port without buffer component.

Pin. 1	Gnd
Pin. 2~4	Reserved
Pin. 5	RS232 RxD signal input to Drive, CMOS/TTL level signal
Pin. 6	RS232 TxD signal output from Drive, CMOS/TTL level signal
Pin. 7	+5VDC output from Drive

⚠ WARNING	Pin. 2 ~ 4 are reserved for factory use and are internally connected. Connecting these pins to external devices may result in permanent damage to servo drive.
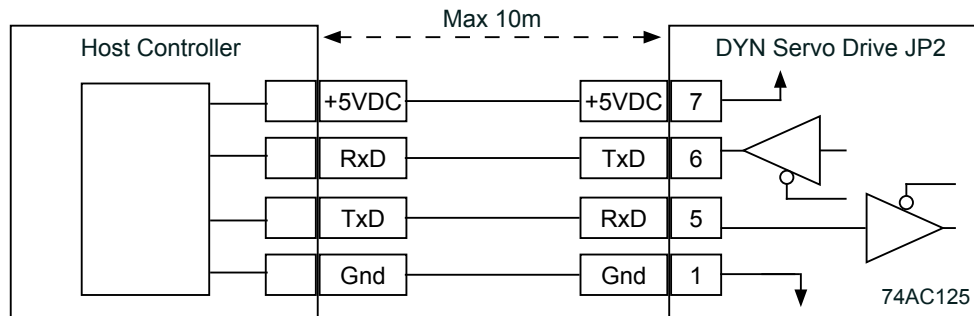
■   Communication Format

Baud Rate	38400
Start/Stop Bit	1
Odd/Even Verify Bit	No
Data	8-bit
	Full Duplex
	Asynchronous (UART)
Voltage	TTL/CMOS

■   Transmission

The DYN servo drive is always under command from host controller. When a function is called, the servo drive will move the servo motor, return a data packet with the requested information, or set a parameter value. Once a complete data packet has been received, the servo drive will not return any confirmation or acknowledgement code. The command motion will be immediately run, requested data will be returned, or new parameter is saved.

The subroutine in Section 7.5A Appendix should be implemented to automatically generate a full data packet. Otherwise, the host controller must ensure each data packet is complete and accurate before transmission.

■   Reception

The DYN servo drive follows same data packet format and structure when returning data. Each data packet is sent one byte at a time consisting of 8 data bits and two start stop bits for a total of 10 bits. Each byte will be sent sequentially until complete packet is sent.

The host controller must process received data in shift register as soon as each byte is transmitted to avoid overflow and garbage data. Alternatively, the receiver shift register buffer must have enough address to store complete packet. The DYN servo drive will send each byte immediately after another, so at 38400 baud, each byte will take approximately 260us to transmit - host controller should read or sample at this rate or faster when receiving data.

7.2.1 Structure

Byte : consists of 8 bits, represented by b7b6b5b4b3b2b1b0 or b[7:0]. b7 is MSB and b0 is LSB, so called little endian. Each packet consists of several bytes, expressed as:

Packet = Bn Bn-1 Bn-2 .... B1 B0
Packet length = n+1, Total n+1 bytes

Bn is start byte, B0 is end byte, similar to the byte structure, Bn is MSB and B0 is LSB as little endian rule.

The integer n varies as the variation of packet length. Functionally, a packet could be expressed as:

Packet = ID + packetLength + functioncode + data + checksum

ID	One byte (Start byte)
packetLength + functioncode	One byte
data	One to four bytes
checksum	One byte

Minimum packet length is 4 bytes, packet length 4 (n=3), 1 data byte.
Maximum packet length is 7 bytes, packet length 7 (n=6), 4 data byte.

Minimum packet length is 4. There is at least one data byte, for some function code that does not require data, this data byte is meaningless, or called dummy byte which can be set to any value [0~127] and does not affect the overall function of that packet.

7.2.2 Features for the byte inside a packet

The start byte takes form of 0xxxxxxx, or MSB is 0, x for 0 or 1. Any other byte except the start byte takes the form of 1xxxxxxx, where x could be 0 or 1. Most significant bit in a byte can be used for determining if it is a packet's start byte or not.

7.2.3 Start byte Bn

The MSB bit of start byte is always zero, the other seven bits are used for the Drive ID number which is set from 0 ~ 63. The ID number can also be assigned through the DMMDRV software.

ID number 127 is reserved for every drive for broadcasting purposes. In other words, 127 is general ID number. ID numbers 64 ~ 126 are internally reserved.

The communicating servo drive must be set to the same ID number to accept and execute data. The drive ID can only be set if the *RS485/232 Net* check box is not checked (in the DMMDRV software).

7.2.4 Bn-1 byte

The Bn-1 byte is used for representing the packet function and packet length.

Bn-1 = 1 b6 b5 b4 b3 b2 b1 b0

The bit b6 and b5 are for the length of packet, expressed as:

b6	b5	Total packet length(=n+1)
0	0	4
0	1	5
1	0	6
1	1	7

The bit b4~b0 are used for the packet function, expressed as:

Function (Sent by host)	b[4:0]	Data (Bytes)	Remarks
Set_Origin	0x00	1(dummy)	Set current position as zero ; See section 7.4.2
Go_Absolute_Pos	0x01	1~4	See section 7.4.1
Make_LinearLine	0x02	1~4	
Go_Relative_Pos	0x03	1~4	See section 7.4.1
Make_CircularArc	0x04	1~4	
Assign_Drive_ID	0x05	1	Assign ID to Drive; See Section 7.6
Read_Drive_ID	0x06	1(dummy)	
Set_Drive_Config	0x07	1	One byte Configuration. See Section 7.3
Read_Drive_Config	0x08	1(dummy)	Read Drive configuration. See Section 7.3
RegisterRead_Drive_Status	0x09	1(dummy)	Ask for Drive status. See Section 7.3
Turn_ConstSpeed	0x0a	1~3	See section 7.4.2
Square_Wave	0x0b	1~3	See section 7.4.2
Sin_Wave	0x0c	1~3	See section 7.4.2
SS_Frequency	0x0d	1~3	See section 7.4.2
General_Read	0x0e	1~4	Read Drive position set
ForMotorDefine	0x0f	1	Internal Function - Not customer accessible
Set_MainGain	0x10	1	
Set_SpeedGain	0x11	1	
Set_IntGain	0x12	1	
Set_TrqCons	0x13	1	
Set_HighSpeed	0x14	1	Set MaxSpd,1~127 ; See section 7.4, 7.5
Set_HighAccel	0x15	1	Set MaxAcl,1~127 ; See section 7.4, 7.5
Set_Pos_OnRange	0x16	1	If |Pset-Pmotor|<= OnRange, then motor on Pos OnRange ;1~127
Set_GearNumber	0x17	2	Gear_Number [500~16,384] ; ; See section 7.4, 7.5
Read_MainGain	0x18	1(dummy)	See section 7.5 Example 11
Read_SpeedGain	0x19	1(dummy)	See section 7.5 Example 11
Read_IntGain	0x1a	1(dummy)	See section 7.5 Example 11
Read_TrqCons	0x1b	1(dummy)	See section 7.5 Example 11
Read_HighSpeed	0x1c	1(dummy)	See section 7.5 Example 11 ; See section 7.4
Read_HighAccel	0x1d	1(dummy)	See section 7.5 Example 11 ; See section 7.4
Read_Pos_OnRange	0x1e	1(dummy)	See section 7.5 Example 11
Read_GearNumber	0x1f	1(dummy)	See section 7.5 Example 11 ; See section 7.4

Functions (Sent by DYN drive)	b[4:0]	Data (Bytes)	Remarks
Not used	0x00 ~ 0x0a		*Do not read or write to these addresses
Is_MainGain	0x10	1	Returns [1:127] unsigned data
Is_SpeedGain	0x11	1	Returns [1:127] unsigned data
Is_IntGain	0x12	1	Returns [1:127] unsigned data
Is_TrqCons	0x13	1	Returns [1:127] unsigned data
Is_HighSpeed	0x14	1	Returns [1:127] unsigned data
Is_HighAccel	0x15	1	Returns [1:127] unsigned data
Is_Drive_ID	0x16	1	Returns [1:127] unsigned data
Is_PosOn_Range	0x17	1	Returns [1:127] unsigned data
Is_GearNumber	0x18	2	
Is_Status	0x19	1	
Is_Config	0x1a	1	
Is_AbsPos32	0x1b	1~4	
Is_TrqCurrent	0x1e	1~4	

Functions 0x10 ~ 0x1e are sent from the DYN drive in response to a function to request data. For example, when Read_MainGain 0x18 is sent to the DYN2 drive, Is_MainGain 0x10 is returned as the function with the Main Gain value as the data. See section 7.5 Example 11.

7.2.5 Bn-2 ~ B1 bytes

Bn-2 ~ B1 (n>2) are used for representing the data in the packet. 7bits of a byte is used for containing the data. The first bit MSB is always 1.

n	Data Range	Remark
3	-64 ~ 63	Only B1 is used
4	-8,192 ~ 8,191	Only B2, B1 are used
5	-1,048,576 ~ 1,048,575	B3, B2, B1 are used
6	-134,217,728 ~ 134,217,727	B4, B3, B2, B1 are used

Minimum packet length is 4. There is at least one data byte, for some function code that does not require data, this data byte is meaningless, or called dummy byte which can be set to any value [0~127] and does not affect the overall function of that packet.

DYN2MS-02F-0116A17

7.2.6 B0 Byte

B0 byte is used for check sum, which is calculated from Bn~B1 as:

$S = B_n + B_{n-1} + B_{n-2} + ... B_1$
$B_0 = 0x80 + Mod(S, 128)$, $B_0 = 0x80 + S - 128*[S/128]$
$B_0 = 128 \sim 255$

After receiving a packet, then calculate $Temp = Mod(S, 128)$, if $Temp = B_0$, there is no error, otherwise there is error during the packet transmission.

Example manual calculation:

> Given: Command to rotate ID=8 motor at 50rpm constant speed
> Packet Length = 4
> n = 3
> $B_3 = 0x08$
> $B_2 = 0x8a$
> $B_1 = 0xb2$
>
> $S = B_3 + B_2 + B_1 = 0x144 = 324$
> $B_0 = 0x80 + Mod(S, 128)$
> $\quad = 0x80 + Mod(324, 128)$
> $\quad = 0x80 + 0x44$
> $B_0 = c4$

DYN2MS-02F-0116A17

Drive configuration such as command input mode (RS232, CW/CCW etc.), alarm status, busy status are described by the two register Config and Status which are stored inside Drives EEPROM and can be read or set through RS232 communication.

▋ Drive Status

Driver status is a byte data, lower 7 bit valid for indicating the Drive status, is it in the state of servo, alarm, on position, or free.

Status = x b6 b5 b4 b3 b2 b1 b0

b0	= 0 :	On position, i.e. |Pset - Pmotor| < = OnRange
b0	= 1 :	motor busy, or |Pset - Pmotor|> OnRange
b1	= 0 :	motor servo
b1	= 1 :	motor free
b4 b3 b2 =	0 :	No alarm
	1 :	motor lost phase alarm, |Pset - Pmotor|>8192(steps), 180(deg)
	2 :	motor over current alarm
	3 :	motor overheat alarm, or motor over power
	4 :	there is error for CRC code check, refuse to accept current command
	5~ 7 :	TBD
b5	= 0 :	means buit in S-curve, linear, circular motion completed; waiting for next motion
b5	= 1 :	means buit in S-curve, linear, circular motion is busy on current motion
b6	:	pin2 status of JP3,used for Host PC to detect CNC zero position or others

▋ Drive Configuration

Drive configuration for communication mode, servo mode etc is expressed by Config.

Config = x b6 b5 b4 b3 b2 b1 b0

b1 b0 =	0 :	RS232 mode
	1 :	CW,CCW mode
	2 :	Pulse/Dir or (SPI mode Optional)
	3 :	Anlog mode
b2	= 0 :	works as relative mode(default) like normal optical encoder
b2	= 1 :	works as absolute position system, motor will back to absolute zero or POS2(Stored in sensor) automatically after power on reset.
b4 b3 =	0 :	Position servo as default
	1 :	Speed servo
	2 :	Torque servo
	3 :	TBD
b5	= 0 :	let Drive servo
b5	= 1 :	let Drive free, motor could be turned freely
b6	:	TBD

The default Config = x0000000, RS232 communication mode, absolute position sensor works as relative mode, position servo, servo enabled. If the bit 5 of Config register is set to be 1, Drive will let motor shaft free (servo disabled).

7.4.1   Point to Point Movement （S-Curve）

Max Acceleration, Max Speed, and Gear Number are important data parameters for generating the S-Curve. The DYN servo drive also applies a smoothing filter to the acceleration profile to generate best S-Curve performance. The S-Curve profile is calculated as the following,

$$\text{Gear Ratio} = \frac{4,096}{\text{GEAR NUMBER}}$$

$$\text{Maximum Motor Speed [rpm]} = \frac{(\text{MaxSpd}+3)*(\text{MaxSpd}+3)}{16} * 12.21 * \text{Gear Ratio}$$

$$\text{Maximum Motor Acceleration [rpm/s]} = \text{MaxAcl} * 635.78 * \text{Gear Ratio}$$

$$\text{Motor Movement Position} = \text{Command Position} * \text{Gear Ratio} * 4$$

Example:

Set parameter	Output
Gear_Num = 4096	Gear Ratio = 1
MaxSpd = 48	Maximum Motor Speed = 1985 rpm
MaxAcl = 30	Maximum Motor Acceleration = 19073 rpm/s
Command Position = 140,000	Motor Movement Position = 560,000 positions

S-Curve:

Acceleration Time = 0.104 s
Distance During Acceleration = 1.72 rev
Constant Speed Travel Time = 0.154 s
Total S-Curve Time = 0.362 s

DYN2MS-02F-0116A17
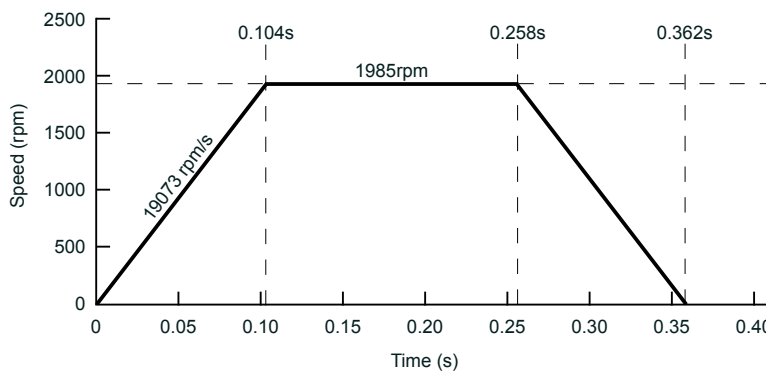
7.4.2 Constant Speed, Square Wave, Sin Wave

■   Turn Constant Speed

Function (Sent by host)	b[4:0]	Data (Bytes)
Turn_ConstSpeed	0x0a	1~3

The servo motor rotates at constant speed according to the rpm speed set by the Data Bytes. The direction of rotation is CW (as viewed from shaft side) for positive speed and CCW for negative speed.

Example:

Set command	Motion
Function = 0x0a (Turn_ConstSpeed) Data = 0x578 (1,400) (use 2 data bytes B2, B1) B2 = **1**000 0101 *MSB must be 1 B1 = **1**111 1000 *MSB must be 1	Servomotor rotates in CW direction (as viewed from shaft side) at 1,400rpm
Function = 0x0a (Turn_ConstSpeed) Data = 0xff88 (-120) (use 2 data bytes B2, B1) 0xff88 = 0x1111 1111 1000 1000 B2 = **1**111 1111 B1 = **1**000 1000	Servomotor rotates in CCW direction (as viewed from shaft side) at 120rpm

■   Square Wave Motion

Function (Sent by host)	b[4:0]	Data (Bytes)
Square_Wave	0x0b	1~3
SS_Frequency	0x0d	1~3

The servo motor makes a square wave motion with instantaneous acceleration and deceleration command. The amplitude is set by the Square_Wave function Data and the frequency is set by the SS_Frequency function Data Bytes. The motion is executed as soon as the Square_Wave function is received. Note that Square_Wave and Sin_Wave shares the same SS_Frequency data value. The square waveform is generated internally within the DYN servo drive.

■   Sine Wave Motion

Function (Sent by host)	b[4:0]	Data (Bytes)
Sin_Wave	0x0c	1~3
SS_Frequency	0x0d	1~3

The servo motor makes a sine wave motion with continuous acceleration and deceleration. The amplitude is set by the Sin_Wave function Data and the frequency is set by the SS_Frequency function Data Bytes. The motion is executed as soon as the Sin_Wave function is received. Note that Sin_Wave and Square_ Wave shares the same SS_Frequency data value. The sine waveform is generated internally within the DYN servo drive.

The DYN servo drives built in S-Curve generator is able to update the target position instantaneously regardless of whether the current command position has completed or not. As soon as a new command position is received, the servo drive immediately updates the servomotor target to the newest position. This function is applicable to both relative (incremental) and absolute positioning for all linear, or arc path profiles.

Without Dynamic Target Position Update DTPU technology, the servo drive must wait until the first, or current position command is completed before executing the next one. This limits the rate at which the motor position can be updated and and can also have detrimental effects on safety for the machine and the operator. With DTPU technology, the servo drive is always under active command from the controller, allowing much faster cycle time and higher universal efficiency.

The servo drive also applies a curved acceleration command to the S-Curve to maintain smoothest servo motor motion. At each S-Curve "transition" point, the normally rigid path is curved into smooth speed transitions.
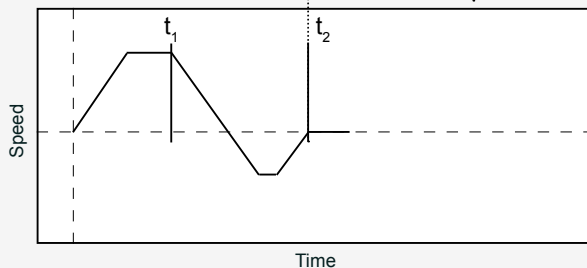
■   Efficiency

Without DTPU
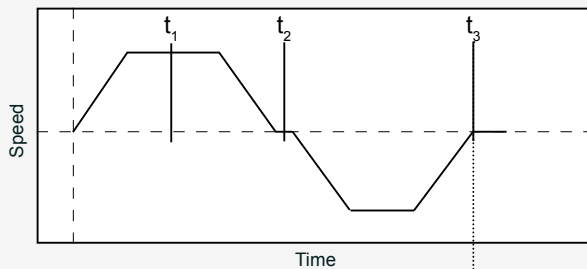
t_1 : Second command position given
t_2 : First position reached, second position executed
t_3 : Second position reached

Time Reduced

With DTPU

t_1 : Second command position given, servo drive immediately targets to second position
t_2 : Second position reached

When the axis is command to a new position, the servo drive immediately updates the target position and generates new S-Curve profile to reach new target position. Without DTPU technology, the axis must first finish its current command before executing the next one, causing a delay in the overall positioning time.

This also allows more flexibility in programming and path planning as the controller no longer needs to wait until a particular movement is finished before calculating the succeeding one. Robotic movements can be controlled and commanded in real-time, significantly simplifying kinematic motion planning requirements on the controller. Machine-level trajectory planning can almost be eliminated.

DYN2MS-02F-0116A17

■   Curved Acceleration

The DTPU algorithm also applies a curved acceleration to maintain smooth motion. At each S-Curve transition point, the acceleration/deceleration is curved at the edges so speed is smoothly changed. This decreases motor vibration. The smoothing is applied relative to total command movement so overall distance and position accuracy is not affected.
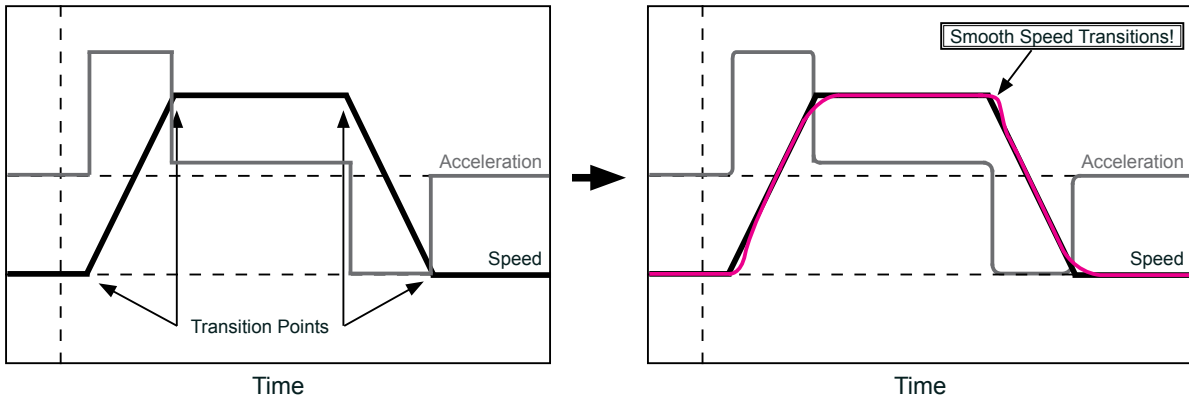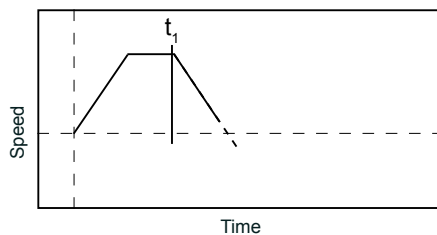

■   Safety

Dynamic Target Position Update DTPU allows the axis to be commanded as soon as a safety hazard or warning is detected. This means protection measures can be executed immediately. Without DTPU, the axis must finish the current positioning before executing protection measures.

Without DTPU
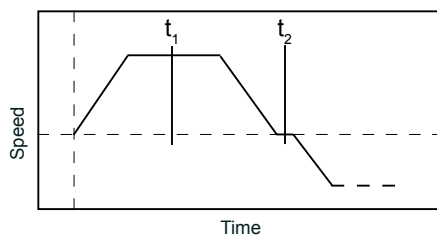
t_1 : Safety warning/hazard detected, axis commanded to retract

t_2 : Current positioning reached, axis commanded to retract

With DTPU

t_1 : Safety warning/hazard detected, axis commanded to retract

t_2 : Axis immediately retracts to safe position

DYN2MS-02F-0116A17

Sent Packet (to DNY drive)		
Function	Function Code	Data (Bytes)
Set_Drive_Config	0x07	1

Bn	Bn-1	Bn-2	B0
0xxxxxxx	10000111	1 b6 b5 b4 b3 b2 b1 b0	1xxxxxxx

Drive configuration setting

Sent Packet (from DYN drive)
None

Sent Packet (to DNY drive)		
Function	Function Code	Data (Bytes)
Read_Drive_Config	0x08	1 (dummy)

Bn	Bn-1	Bn-2	B0
0xxxxxxx	10001000	1 b6 b5 b4 b3 b2 b1 b0	1xxxxxxx

Dummy bits

Received Packet (from DYN drive)		
Is_Config	0x1a	1

Bn	Bn-1	Bn-2	B0
0xxxxxxx	10011010	1 b6 b5 b4 b3 b2 b1 b0	1xxxxxxx

Packet Length = 4   Drive configuration data
Function = 0x1a

Sent Packet (to DNY drive)		
Function	Function Code	Data (Bytes)
Read_Drive_Status	0x09	1 (dummy)

Bn	Bn-1	Bn-2	B0
0xxxxxxx	10001000	1 b6 b5 b4 b3 b2 b1 b0	1xxxxxxx

Dummy bits

Received Packet (from DYN drive)		
Is_Status	0x19	1

Bn	Bn-1	Bn-2	B0
0xxxxxxx	10011001	1 b6 b5 b4 b3 b2 b1 b0	1xxxxxxx

Packet Length = 4   Drive status data
Function = 0x19

■ EXAMPLE 4

Condition:
Make 3th axis motor move -120(steps) from right now position, ID = 3.

Method:
-120 = 0x88 = 0xff88 < -63,Two byte data. 0xff88 = 0x1111 1111 1000 1000: Lower 7bits = 000 1000 = 0x08   Higher 7bits = 0111 1111 = 0x7f Use function Go_Relative_Pos(=0x03), Packet Length = 5. B4 = 0x03; B3 = 0x80 +(PacketLength-4)*32 + Go_Relative_Pos = 0x80 +0x04 =0xa3. B2 = 0x80 + 0x7f = 0xff B1 = 0x80 + 0x08 = 0x88 S = B4 + B3 + B2 + B1 = 0x03 + 0xa3 + 0xff + 0x88 = 0x22d B0 = 0x80 + Mod(S , 128) = 0x80 + 0x2d = 0xad

■ EXAMPLE 5

Condition:
Make 2th axis motor turn at 60rpm, ID = 2.

Method:
Speed is 60, One Byte data is enough, 60 = 0x3c. Packet Length = 4. B3 = 0x02; B2 = 0x80 +(PacketLength-4)*32 + Turn_ConstSpeed = 0x80 +0x0a = 0x8a B1 = 0x80 + 0x3c = 0xbc S = B3 + B2 + B1 = 0x02 + 0x8a + 0xbc B0 = 0x80 + Mod(S , 128) = 0xc8

■ EXAMPLE 6

Condition:
Make 2th axis motor turn at -60rpm, ID = 2. Speed is -60 = 0xc4 = 0x1100 0100 > -63, One byte data 7bits = 0x0100 0100 = 0x44. Packet Length = 4.

Method:
B3 = 0x02; B2 = 0x80 +(PacketLength-4)*32 + Turn_ConstSpeed = 0x80+0x40+0x0a = 0x8a B1 = 0x80 + 0x44 = 0xc4 S = B3 + B2 + B1 = 0x02 + 0x8a + 0xc4 = 0x150 B0 = 0x80 + Mod(S , 128) = 0x80 + 0x50 = 0xd0

■ EXAMPLE 7

Condition:

Make a line on X-Y Plane
Suppose right now position for three motors are(X0,Y0,Z0) = (0,0,0),
and the End point of straight line is (X1,Y1,Z1) = (100,200,0)

Method:

Always use General ID = 0x7f
The Feedrate = 3, could be from 1~127
Global_Func = (char)Make_LinearLine = 0x02;

Then send four packets to the Drives as:
Send_Package(ID,X1 - X0), i.e. Send_Package(0x7f,100)
Send_Package(ID,Y1 - Y0), i.e. Send_Package(0x7f,200)
Send_Package(ID,Z1 - Z0), i.e. Send_Package(0x7f,0)
Send_Package(ID,FeedRate),i.e. Send_Package(0x7f,3)

After the X-Y-Z three Drives received all four packets, they will start to move until the meet the end point of (X1,Y1,Z1). Three motors will meet (X1,Y1,Z1) at the same time.

During the linear or circular interpolation motion, the Read_Drive_Status(=0x09) can used to read Drives status register to check whether b5 = 0 or not, b5 = 0 means the coordinated motion be finished.

Send_Package(ID,Y1 - Y0) is the subroutine in the SAMPLE CODE, it will generate a packet as above examples.

■ EXAMPLE 8

Condition:

Make a circular arc on X-Y Plane

Suppose right now position for three motors are(X0,Y0) = (0,0), and the End point of arc is (X1,Y1) = (200,0) in CW direction. It is easy to know the center of arc is (Xc,Yc) = (100,0)

Method:

The Feedrate = 1, could be from 1~127>0, because in CW direction otherwise be negative value.
The planeNumber = 0 because it is in X-Y plane
TwoBytes = (PlaneNumber<<8) | FeedRate = 0*256 + 1 = 1
Use General ID = 0x7f
Global_Func = (char)Make_CircularArc = 0x04;

Then send five packets to the Drives as:
Send_Package(ID,X0 - Xc), i.e. Send_Package(0x7f,-100)
Send_Package(ID,Y0 - Yc), i.e. Send_Package(0x7f,0)
Send_Package(ID,X1 - Xc), i.e. Send_Package(0x7f,100)
Send_Package(ID,Y1 - Yc), i.e. Send_Package(0x7f,0)
Send_Package(ID,TwoBytes),i.e. Send_Package(0x7f,1)

After the X-Y-Z three Drives received all four packets, Only two of three motors will move and finally will meet (X1,Y1) at the same time. During the linear or circular interpolation motion, the Read_Drive_Status (=0x09) can used to read Drives status register to check whether b5 = 0 or not, b5 = 0 means the coordinated motion be finished.

Two equal half arcs must be made to make a circle.

The following three examples makes use of the sample code in *Section 7.7A Appendix : C++ Code for Serial Communication Protocol*. All contents of the sample code must be copied to the program.

■ EXAMPLE 9

Condition:
Read servo motor absolute position
Method:
Call ReadMotorPosition32() subroutine function Motor position stored in Motor_Pos32 variable as: Motor_Pos32 = (long) [-2^27 : 2^27-1] = [-134,217,728 : 134,217,727]

■ EXAMPLE 10

Condition:
Read servo motor torque current
Method:
Call ReadMotorTorqueCurrent() subroutine function Motor torque current stored in MotorTorqueCurrent variable as: MotorTorqueCurrent = (short) [-2^15 : 2^15-1] = [−32,767 : 32,766] MotorTorqueCurrent represents a relative number according to the RMS current output by servo drive. This value is different between each servo motor capacity and varies between the DYN2 and DYN4 servo drive. The customer can measure the change in MotorTorqueCurrent variable to monitor relative current draw. Use servo motor torque constant specification to calculate torque output.

■ EXAMPLE 11

Condition:
Read servo drive Main Gain parameter
Method:
Call ReadMainGain() subroutine function DYN drive Main Gain stored in MainGain_Read variable Use the same subroutine format for all Parameter Read functions 0x18~ 0x1f.

DYN2MS-02F-0116A17

Several Drives can be connected by RS485 after every Drive on the RS485 net have been designated an individual, or broadcasting ID number.

The RS485 check box must be checked if RS485 network is used which means there are at least two or more Drive on the net, then every servo drive status and configuration can be read or set according to the ID number on the servo setting dialog box. The ID number cannot be assigned to a particular Drive if RS485 network is connected.

The Servo Drive ID number CAN ONLY BE SET when there is only ONE drive connected, then assigned a new ID number to that drive without checking the *RS485/232 Net* **check box (in the DMMDRV software).**

The RS485 network is a serial network, if there is a packet in the network, one Drive will receive it first, if the packet's ID number is the same as the Drives, that packet will be received and processed by the Drive, otherwise that packet will be relayed to the next Drive.

The Drive ID is contained in the first byte of the packet. When a packet is received, the drive only reads the first byte, it will receive if ID is correct and relay to next drive if ID does not match. Data flow on the serial RS485 net is very fast and efficient.

Every drive has a RS485NET node which contains a RS485 buffer such as LTC491.
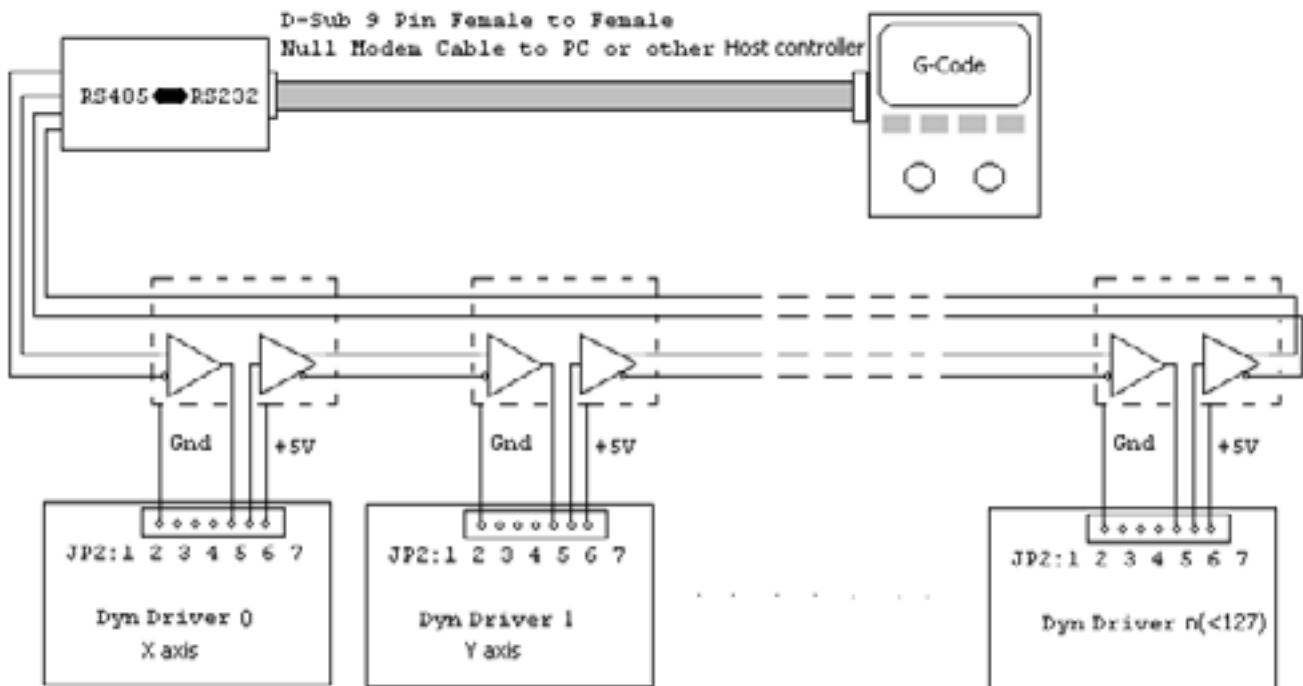
The following code shows an example to generate a data packet and call functions in RS232 serial protocol.

Note: in the description of RS232 communication protocol above (Section 7), the last byte of packet is always B0, but in the code of below, the first byte is always B0.

```
#define Go_Absolute_Pos                 0x01
#define Is_AbsPos32                     0x1b
#define General_Read                    0x0e
#define Is_TrqCurrent                   0x1e
#define Read_MainGain                   0x18
#define Is_MainGain                     0x10
        char InputBuffer[256];                              //Input buffer from RS232,
        char OutputBuffer[256];                             //Output buffer to RS232,
        unsigned char InBfTopPointer,InBfBtmPointer;//input buffer pointers
        unsigned char OutBfTopPointer,OutBfBtmPointer;//output buffer pointers
        unsigned char Read_Package_Buffer[8],Read_Num,Read_Package_Length,Global_Func;
        unsigned char MotorPosition32Ready_Flag, MotorTorqueCurrentReady_Flag, MainGainRead_Flag;
        long Motor_Pos32;
        int MotorTorqueCurrent, MainGain_Read;

void DlgRun::ReadPackage()
{
        unsigned char c,cif;

        ReadRS232Port();                                // Include customer code to read from serial port

        while(There is data in the customer hardware RS232 receiving Buffer)
        {
                InputBuffer[InBfTopPointer] = HardwaerRS232ReceiveBuffer;        //Load InputBuffer with received packets
                InBfTopPointer++;
        }

        while(InBfBtmPointer!=InBfTopPointer)
        {
                c = InputBuffer[Comm.InBfBtmPointer];
                InBfBtmPointer++;
                cif = c&0x80;
                if(cif==0)
                {
                        Read_Num = 0;
                        Read_Package_Length = 0;
                }
                if(cif==0||Read_Num>0)
                {
                        Read_Package_Buffer[Read_Num] = c;
                        Read_Num++;
                        if(Read_Num==2)
                        {
                                cif = c>>5;
                                cif = cif&0x03;
                                Read_Package_Length = 4 + cif;
                                c = 0;
                        }
                        if(Read_Num==Read_Package_Length)
                        {
                                Get_Function();
                                Read_Num = 0;
                                Read_Package_Length = 0;
                        }
                }
        }
}
```

C++ Code for Serial Communication - Page 1

DYN2MS-02F-0116A17

```cpp
void DlgRun::Get_Function(void)
{
        char ID, ReceivedFunction_Code, CRC_Check;
        ID = Read_Package_Buffer[0]&0x7f;
        ReceivedFunction_Code = Read_Package_Buffer[1]&0x1f;
        CRC_Check = 0;
        for(int i=0;i<Comm.Read_Package_Length-1;i++)
        {
                CRC_Check += Read_Package_Buffer[i];
        }
        CRC_Check ^= Read_Package_Buffer[Comm.Read_Package_Length-1];
        CRC_Check &= 0x7f;
    if(CRC_Check!= 0){
        //MessageBox("There is CRC error!") - Customer code to indicate CRC error
    }
    else
    {
    switch(ReceivedFunction_Code){
        case  Is_AbsPos32:
                Motor_Pos32 = Cal_SignValue(Read_Package_Buffer);
                MotorPosition32Ready_Flag = 0x00;
                break;
        case  Is_TrqCurrent:
                MotorTorqueCurrent = Cal_SignValue(Read_Package_Buffer);
                MotorTorqueCurrentReady_Flag = 0x00;
                break;
        case  Is_MainGain:
                MainGain_Read = Cal_SignValue(Read_Package_Buffer);
                MainGainRead_Flag = 0x00;
                break;
        default:;
        }
    }
}


/*Get data with sign - long*/
long DlgRun::Cal_SignValue(unsigned char One_Package[8])
{
        char Package_Length,OneChar,i;
        long Lcmd;
        OneChar = One_Package[1];
        OneChar = OneChar>>5;
        OneChar = OneChar&0x03;
        Package_Length = 4 + OneChar;
        OneChar = One_Package[2];               /*First byte 0x7f, bit 6 reprents sign */
        OneChar = OneChar << 1;
        Lcmd = (long)OneChar;                   /* Sign extended to 32bits */
        Lcmd = Lcmd >> 1;
        for(i=3;i<Package_Length-1;i++)
        {
                OneChar = One_Package[i];
                OneChar &= 0x7f;
                Lcmd = Lcmd<<7;
                Lcmd += OneChar;
        }
        return(Lcmd);                           /* Lcmd : -2^27 ~ 2^27 - 1 */
}
```

C++ Code for Serial Communication - Page 2

60

```cpp
//***************** Every Robot Instruction ******************
// Send a package with a function by Global_Func
// Displacement: -2^27 ~ 2^27 - 1
// Note: in the description of RS232 communication protocol above (Section 7), the last byte of packet is    //
always B0, but in the code of below, the first byte is always B0.

void DlgRun::Send_Package(char ID , long Displacement)
{
        unsigned char B[8],Package_Length,Function_Code;
        long TempLong;
        B[1] = B[2] = B[3] = B[4] = B[5] = (unsigned char)0x80;
        B[0] = ID&0x7f;
        Function_Code = Global_Func & 0x1f;
        TempLong = Displacement & 0x0fffffff;              //Max 28bits
        B[5] += (unsigned char)TempLong&0x0000007f;
        TempLong = TempLong>>7;
        B[4] += (unsigned char)TempLong&0x0000007f;
        TempLong = TempLong>>7;
        B[3] += (unsigned char)TempLong&0x0000007f;
        TempLong = TempLong>>7;
        B[2] += (unsigned char)TempLong&0x0000007f;
        Package_Length = 7;
        TempLong = Displacement;
        TempLong = TempLong >> 20;
        if(( TempLong == 0x00000000) || ( TempLong == 0xffffffff))
        {//Three byte data
                B[2] = B[3];
                B[3] = B[4];
                B[4] = B[5];
                Package_Length = 6;
        }
        TempLong = Displacement;
        TempLong = TempLong >> 13;
        if(( TempLong == 0x00000000) || ( TempLong == 0xffffffff))
        {//Two byte data
                B[2] = B[3];
                B[3] = B[4];
                Package_Length = 5;
        }
        TempLong = Displacement;
        TempLong = TempLong >> 6;
        if(( TempLong == 0x00000000) || ( TempLong == 0xffffffff))
        {//One byte data
                B[2] = B[3];
                Package_Length = 4;
        }
        B[1] += (Package_Length-4)*32 + Function_Code;
        Make_CRC_Send(Package_Length,B);
}
```

C++ Code for Serial Communication - Page 3

61

```cpp
void DlgRun::Make_CRC_Send(unsigned char Plength,unsigned char B[8])
{
        unsigned char Error_Check = 0;
        for(int i=0;i<Plength-1;i++)
        {
                OutputBuffer[OutBfTopPointer] = B[i];
                OutBfTopPointer++;
                Error_Check += B[i];
        }
        Error_Check = Error_Check|0x80;
        OutputBuffer[OutBfTopPointer] = Error_Check;
        OutBfTopPointer++;

        while(OutBfBtmPointer != OutBfTopPointer)
        {
                RS232_HardwareShiftRegister = OutputBuffer[OutBfBtmPointer];
                SendRS232Port();                          // Include customer code to send to RS232 port
                OutBfBtmPointer++;                        // Change to next byte in OutputBuffer to send
        }
}

void DlgRun::ReadMotorTorqueCurrent(void)
{/*Below are the codes for reading the motor torque current */

                                                    //Read motor torque current
        char ID = 0;                                //Suppose read 0 axis motor
        Global_Func = General_Read;
        Send_Package(ID , Is_TrqCurrent);

                //Function code is General_Read, but one byte data is : Is_TrqCurrent
                //Then the drive will return a packet, Function code is Is_TrqCurrent
                //and the data is 16bits Motor torque current.


        MotorTorqueCurrentReady_Flag = 0xff;
        While(MotorTorqueCurrentReady_Flag != 0x00)
        ReadPackage();

                //MotorTorqueCurrentReady_Flag is cleared inside ReadPackage() or inside
                //Get_Function() exactly after the MotorTorqueCurrent is updated.
}
```

C++ Code for Serial Communication - Page 4

```
void DlgRun::ReadMotorPosition32(void)
{/*Below are the codes for reading the motor shaft 32bits absolute position */

                                            //Read motor 32bits position
          char ID = 0;                      //Suppose read 0 axis motor
          Global_Func = General_Read;
          Send_Package(ID , Is_AbsPos32);

                    // Function code is General_Read, but one byte data is : Is_AbsPos32
                    // Then the drive will return a packet, Function code is Is_AbsPos32
                    // and the data is 28bits motor position32.

          MotorPosition32Ready_Flag = 0xff;
          While(MotorPosition32Ready_Flag != 0x00)
          ReadPackage();

                    // MotorPosition32Ready_Flag is cleared inside ReadPackage() or inside
                    // Get_Function() exactly after the Motor_Pos32 is updated.
}


void MoveMotorToAbsolutePosition32(char MotorID,long Pos32)
{
          char Axis_Num = MotorID;
          Global_Func = (char)Go_Absolute_Pos;
          Send_Package(Axis_Num,Pos32);
}

void ReadMainGain(char MotorID)
{
          char Axis_Num = MotorID;
          Global_Func = (char)Read_MainGain;
          Send_Package(Axis_Num, Is_MainGain);

          MainGainRead_Flag = 0xff;
          while(MainGainRead_Flag != 0x00)
          {
              ReadPackage();
          }
}
```

C++ Code for Serial Communication - Page 5

63

```
void main(void)
{
        /* (1)  Move motor 2 to absolute position of 321,456 - Method 1*/
        char Axis_Num = 2;
        Global_Func = (char)Go_Absolute_Pos;
        long pos = 321456;
        Send_Package(Axis_Num,Pos);

        /* (2)  Move motor 2 to absolute position of 321,456 - Method 2 - Using subroutine function*/
        MoveMotorToAbsolutePosition32(2,321456);

        /* (3)  Code for reading the motor shaft 32bits absolute position - Method 1
                This method uses a while delay to wait for Send_Package() function to complete
        */
        int i;
        InBfTopPointer = InBfBtmPointer = 0;                //reset input buffer pointers
        OutBfTopPointer = OutBfBtmPointer = 0;              //reset output buffer pointers

        for(i=0;i<8;i++)
                Read_Package_Buffer[i] = 0;

        Read_Num = Read_Package_Length = 0;

        //Reading motor 32bits position
        char ID = 0; //Suppose read 0 axis motor
        Global_Func = General_Read;
        Send_Package(ID , Is_AbsPos32);

        while(i<10000)                          //10~20ms waiting
        {
                i++;
        }

        ReadPackage();                          //Motor absolute position stored in Motor_Pos32 variable

        /* (4)  Reading the motor shaft 32bits absolute position - Method 2 using subroutine function*/
        ReadMotorPosition32();                  //Motor absolute position stored in Motor_Pos32 variable

        /* (5)  Reading the motor current using subroutine function*/
        ReadMotorTorqueCurrent();    //Motor torque current stored in MotorTorqueCurrent variable

        /* (6)  Reading the main gain of 8th axis servo drive using subroutine function*/
        ReadMainGain(8);             //Main Gain stored in MainGain_Read variable
}
```

DYN2MS-02F-0116A17

Sample Code Notes:

(1) The sample code uses a ring buffer structure to input and output data packet bytes. Two separate ring buffers are using in the code as *char InputBuffer[256]* and *char OutputBuffer[256]*.

Two position pointers are used in each buffer structure to index the data inside the buffer structure. For example, when a data packet is received from the servo drive, each byte received is sequentially saved into the InputBuffer with the InBfTopPointer incremented each time. This is done until the host hardware RS232 receiver buffer is empty, meaning all packet bytes have been read and stored. Data is processed as first-in-first-out (FIFO) queue and starts at the index of InBfBtmPointer. InBfBtmPointer is incremented each time a byte is processed until InBfBtmPointer=InBfTopPointer, meaning all packet bytes have been processed.

C++ Code for Serial Communication - Page 7

DYN2MS-02F-0116A17

8   Modbus RTU (RS485) Communication

The DYN2-□□B6S-00 servo drive models are compatible with Modbus RTU communication over 2-Wire RS485.

Please refer to the following manual for Modbus communication specification:

Document Number DYNMB1-BL1645	DYN AC Servo Drive Modbus RTU Specification

9   CAN Communication

The DYN2-□□B6S-00 servo drive models are compatible with CAN 2.0A specification. The data frame format is a proprietary DYN servo drive format with efficient data packaging and high transmission rates up to 1Mbit/s for fastest cycle time.

Please refer to the CAN communication manual for detailed specificaitons.

DYN servo drive CAN Protocol Data Framing:
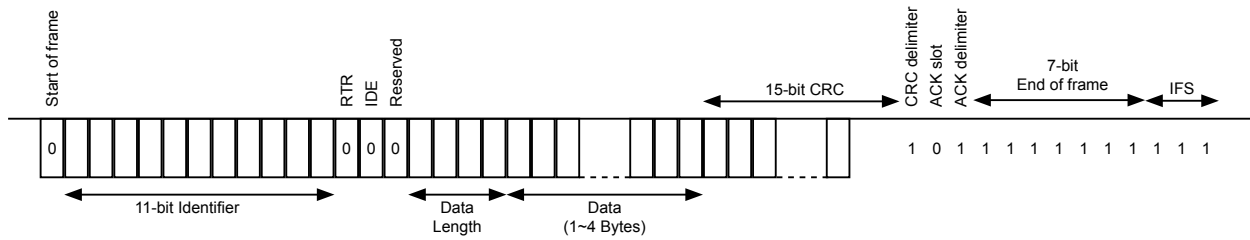
11-bit Identifier Consists both Drive ID and Command Function Code:
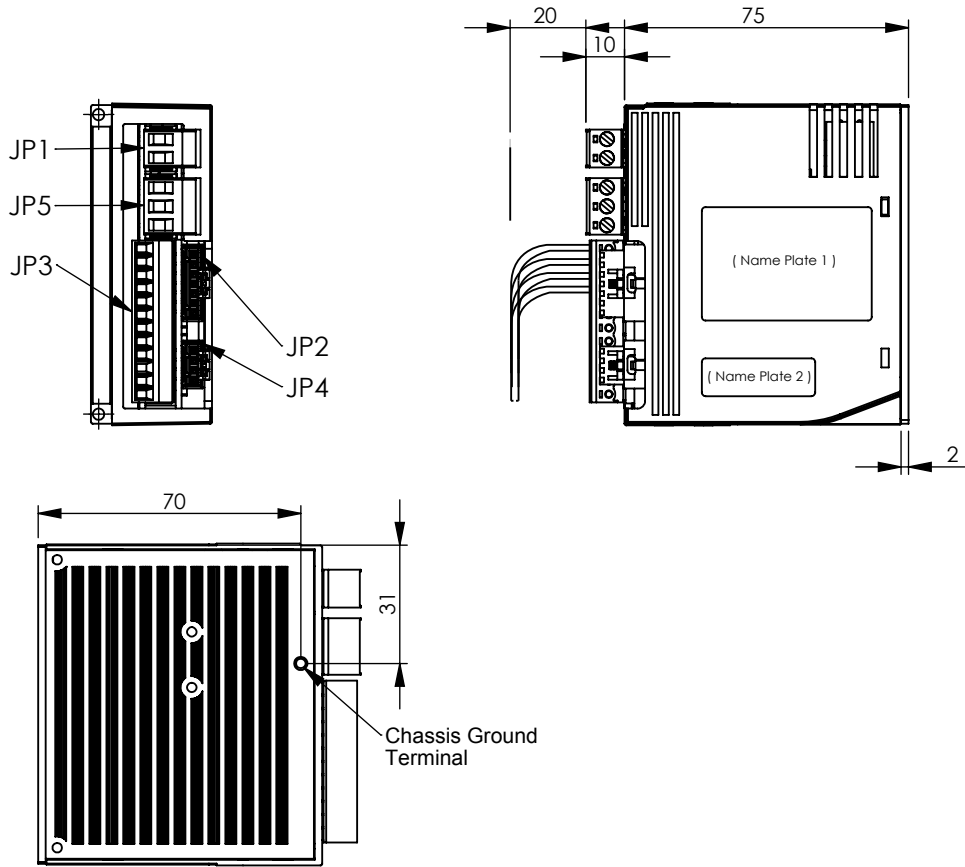
b4~b0 = 5-bit Function Code

b5~b10 = Drive ID 0~64
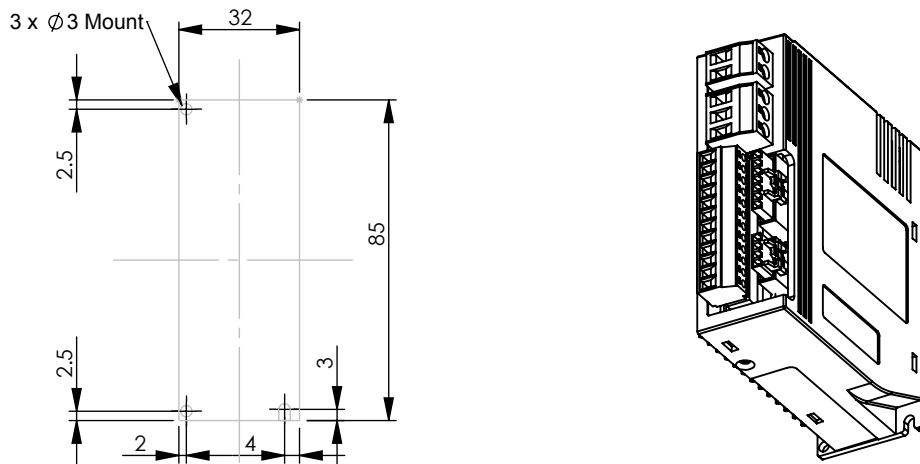0 = Broadcast

Function Code:

#	CAN Command	5-bit Function Code	Data Length (Bytes)
0	Set_Origin	0x00	0
1	Go_Absolute_Pos_PTP	0x01	1~4
2	Make_LinearLine	0x02	1~4
3	Go_Relative_Pos_PTP	0x03	1~4
...	...	...	...

DYN2MS-02F-0116A17

APPENDIX A - SERVO DRIVE DIMENSIONS

♦ Exterior Dimensions

Chassis Ground Terminal

♦ Mounting (as viewed from rear)

3 x ∅3 Mount

DYN2MS-02F-0116A17
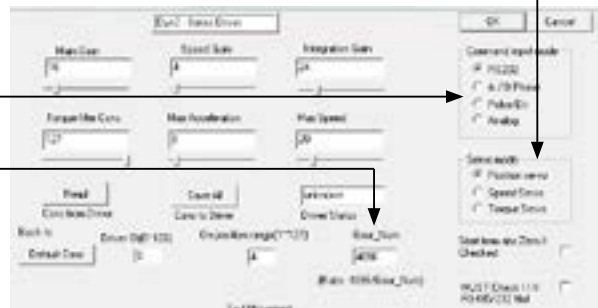
■   Position Servo Mode - Ball Screw

1. Connect encoder feedback and motor power cable from servo drive to servo motor.
2. Connect RS232 tuning cable from servo drive JP2 to controller PC.
3. Power ON servo drive.
4. Open Windows Device Manager - Locate COM Port Number of RS232 tuning cable.

5. Open DMMDRV program.
6. Select COMSET --> COM PORT. Press "Change Port" until RS232 tuning cable COM port number selected. Press "ok".
7. Select ServoSetting --> DYN2-DRIVER. *Setting drive parameters and mode* main screen will open.
8. Press "Read" to read out the factory default or current setting of the servo drive. At any time, pressing "Save All" will save the parameters into the servo drive.
9. Under Servo Mode, select "Position Servo".

10. Under command input mode, select "Pulse/Dir", "A/B Phase", or "CW/CCW".

11. Set GEAR_NUM parameter according to ball screw pitch and target travel speed.
    <u>Example:</u>
    Ball screw pitch = 10mm
    Reduction = 2:1
    Target Speed = 15m/min
    Rated Motor Speed = 3,000rpm = 50rev/s
    Controller Pulse Output Frequency = 100kHz = 100,000pulse/s

    3,000rpm / 2 = 1,500rpm after reduction
    1,500rpm * 10mm = 15,000mm/min = 15m/min

    100,000pulse/s / 50rev/s = 2,000pulse/rev
    2,000pulse/rev / 4 = 500
    GEAR_NUM = 500

12. Tune Gain and OnPosition Range according to machine and operation requirements.
13. Click "Save All" when finished adjustments.
14. The servo drive is ready to accept position pulse commands.

DYN2MS-02F-0116A17

Warranty and Liability

██   Warranty

Products from DMM Technology Corp. are supported by the following warranty.
- 1-year from the date of product received by customer or 14 months from the month of original invoice.

Within the warranty period, DMM Technology Corp. will replace or repair any defective product free of charge given that DMM Technology Corp. is responsible for the cause of the defect. This warranty does not cover cases involving the following conditions:
- The product is used in an unsuitable or hazardous environment not outlined in this manual, resulting in damages to the product.
- The product is improperly handled resulting in physical damage to the product. Including falling, heavy impact, vibration or shock.
- Damages resulting from transportation or shipping after the original factory delivery.
- Unauthorized alterations or modifications have been made to the product.
- Alterations have been made to the Name Plate of the product
- Damages resulting in usage of the product not specified by this manual.
- Damages to the product resulting from natural disasters.
- The product has cosmetic alterations.
- The product does not conform to the original factory manufactured standards due to unauthorized modifications.

██   Liability

Use, operation, handling and storage of the DYN2 AC Servo Drive is solely responsible by the customer. Any direct or indirect commercial loss, commercial profit, physical damage or mechanical damage caused by the DYN2 AC Servo Drive is not responsible by DMM Technology Corp. The features and functionality of the product should be used with full discretion by the customer.

DYN2MS-02F-0116A17

Product and Manual Disclaimer

■ Disclaimer

DMM Technology Corp. constantly strive to improve it's products performance and reliability. The contents of this manual outlines the latest features and specifications of the DYN2 AC Servo Drive and may be changed at any time to reflect corrections, improvements or changes to the product or information in this manual.

■ Manual Revision History

Version	Manual Number	Details	Date
A1	DYN2MS-01E-0814A1	Release Version	August 2014
A1.4	DYN2MS-01E-0115A14	- Updated pulse timing. Section 4.1 - Updated mounting dimensions	January 2015
A1.5	DYN2MS-01E-0315A15	- Included RS232 Communication - Corrections for error	March 2015
A1.7	DYN2MS-02F-0116A17	- Added DTPU Section - Revised Section 7 - Added Receptive Tuning Information - Corrections for error - Layout Changes - Added DC Bus connection diagram - Added regenerative circuit - Adaptive tuning section revised	January 2016
A1.8	DYN2MS-14F-0917A18	- Updated CE certification - Manual updated with B and C type servo drive	September 2017

DYN2 Series
AC Servo Drive

TYPE A - GENERAL PURPOSE PULSE / ANALOG / RS232
TYPE B - MODBUS
TYPE C - CAN

Specification Manual