NANOMOTION

A Johnson Electric Company

FlexDC™

Software User Manual

FLDC458001-01 REV:B **August 29, 2012**

Nanomotion Ltd. POB 623, Yokneam 20692, Israel
Tel: 972-73-2498000 Fax: 972-73-2498099
Web Site: www.nanomotion.com
E-mail: nano@nanomotion.com

Copyright

Limited Warranty

Nanomotion (hereinafter NM) warrants the product (other than software) manufactured by it to be free from defects in material and workmanship for a period of time of one year (except those parts normally considered as consumable/expendable components such as motor conditioning brushes). The warranty commences thirty (30) days from the date of shipment.

NM warrants those parts replaced under warranty for a period equal to the remaining warranty coverage of the original part.

NM's sole and exclusive obligation under this warranty provision shall be to repair, or at its sole option exchange defective products or the relevant part or component, but only if : (i) the Purchaser reports the defect to NM in writing and provides a description of the defective product and complete information about the manner of its discovery within ten (10) days of its discovery; (ii) NM has the opportunity to investigate the reported defect and determines that the defect arises from faulty material, parts or workmanship; and (iii) the Purchaser returns the affected product to a location designated by NM. These provisions constitute the exclusive remedy of the Purchaser for product defects or any other claim of liability in connection with the purchase or use of NM products.

This warranty policy applies only to NM products purchased directly from NM or from an authorized NM distributor or representative.

This warranty shall not apply to (i) products repaired or altered by anyone other than those authorized by NM; (ii) products subjected to negligence, accidents or damage by circumstances beyond NM control; (iii) product subjected to improper operation or maintenance (i.e. operation not in accordance with NM Installation Manuals and/or instructions) or for use other than the original purpose for which the product was designed to be used.

The warranty stands only when the motors are used with the NM drivers/ amplifiers.

NM shall not in any event have obligations or liabilities to the Purchaser or any other party for loss of profits, loss of use or incidental, increased cost of operation or delays in operation, special or consequential damages, whether based on contract, tort (including negligence), strict liability, or any other theory or form of action, even if NM has been advised of the possibility thereof, arising out of or in connection with the manufacture, sale, delivery, use, repair or performance of the NM products. Without limiting the generality of

Nanomotion Ltd. POB 623, Yokneam 20692, Israel
Tel: 972-73-2498000 Fax: 972-73-2498099
Web Site: www.nanomotion.com
E-mail: nano@nanomotion.com

the preceding sentence, NM shall not be liable to the Purchaser for personal injury or property damages.

Patent Information

Nanomotion products are covered under one or more of the following registered or applied for patents.

5,453,653; 5,616,980; 5,714,833; 111597; 5,640,063; 6,247,338; 6,244,076; 6,747,391; 6,661,153; 69838991.3; 6,384,515; 7,119,477; 7,075,211; 69932359.5;1186063; 7,211,929; 69941195.5; 1577961; 4813708; 6,879,085; 6,979,936; 7,439,652; 7061158 ;1800356; 1800356; 1800356; 2007-533057 (pending); 2011-093431 (pending); 7,876,509; 10-2007-7009928 (pending); 200780019448.6 ; 7713361.9 (pending); 12/294,926 (pending); GB2008000004178 (pending); GB2009000003796 (pending); 12/398,216 (pending); GB2446428; 12/517,261 (pending); 08702695.1 (pending); 10-2009-7017629 (pending); 12/524,164 (pending); 12/581,194 (pending)

Revision History

Revision	Release date	Details
00/A	Nov. 2008	New release
01/A	Jan. 2009	Conjoin all SW documentation into “FlexDC SW User Manual”. Add “FlexDC Installation and Operation Sequence” schema.
01/B	Jul 2009	The Bytes tables edited for consistency.
NA	Aug. 2012	Administrative change – added patent information section in front matter.

Table of Contents

Nanomotion Ltd. POB 623, Yokneam 20692, Israel
Tel: 972-73-2498000 Fax: 972-73-2498099
Web Site: www.nanomotion.com
E-mail: nano@nanomotion.com

List of Figures

Nanomotion Ltd. POB 623, Yokneam 20692, Israel
Tel: 972-73-2498000 Fax: 972-73-2498099
Web Site: www.nanomotion.com
E-mail: nano@nanomotion.com

List of Tables

Part I – Preface

1 Preface

1.1 Overview

FlexDC Motion Controller — Nanomotion's dual-axis controller driver. The FlexDC is offered in either single or dual axis configurations. It is a stand alone system, which operates with a standard power supply of 100-240 VAC, 50-60 Hz (for FlexDC technical data, refer to "FlexDC User Manual").

This "FlexDC Software User Manual" is divided into six parts. Every part contains detailed information of the subject it is discussing. The user can easily navigate through the manual using the embedded links in every chapter.

1.2 FlexDC Software User Manual Structure

Part I– Preface.
Chapter 1 – Preface.
Part II– FlexDC Software and Commands Reference. This part focuses on the FlexDC communication syntax, protocols, including response to commands clauses and errors.
Chapter 2 – Commands Syntax and Protocols. This chapter provides full information over the supported software features of the product, and gives a user technical reference for each keyword supported by a communication protocol.
Chapter 3 – Motion Modes. This chapter describes the various motion modes that are supported by the FlexDC.
Chapter 4 – The Control Filter. This chapter describes the Linear Filters equations and Non-linear Algorithms.
Chapter 5 – Faults Protections and Limits. This chapter provides detailed description of faults, protections and the controller's response in each case.

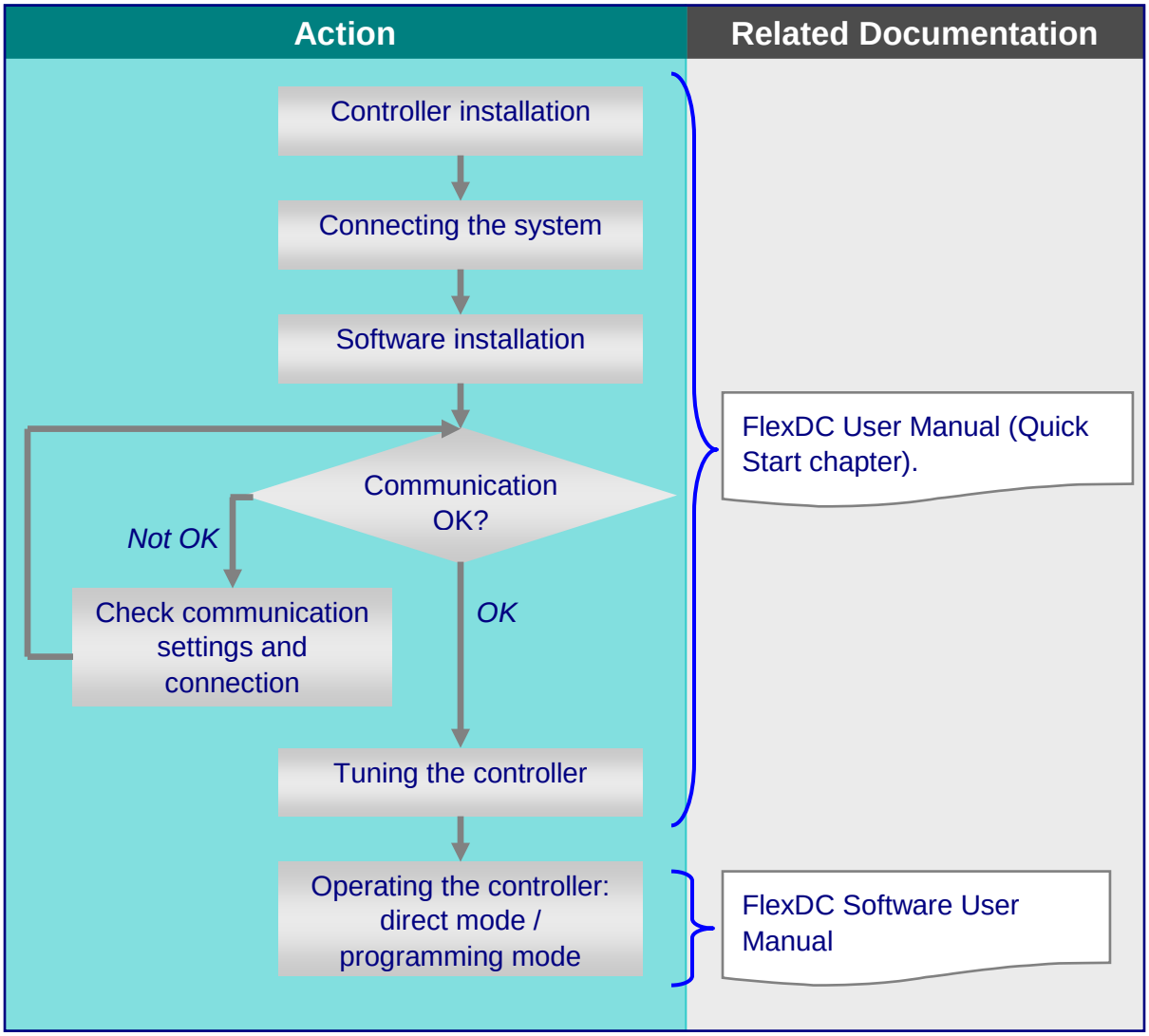
Document Structure (continued)

Chapter 6 – Advanced Features. This chapter describes the FlexDC advanced features such as "Data Recording", "Advanced Encoder Interfaces" and others.
Chapter 7 – Keywords Reference. This chapter describes the keywords supported by the flexDC firmware.
Chapter 8 – Communication and Program Error Codes. This chapter lists all possible communication and program Error Codes (EC) supported by the FlexDC firmware.
Part III– FlexDC Macro Language. This part describes the FlexDC macro-programming language and environment.
Chapter 9 – Introduction.
Chapter 10 – FlexDC Macro Engine.
Chapter 11 – FlexDC Low-Level Macro Program
Chapter 12 –Integrated Development Environment. This chapter covers all the details of the Nanomotion Shell Application, related to macro programming support for the development, downloading, and debugging of FlexDC macros.
Chapter 13 – The IDE Pre-Compiler Support. This chapter discusses the built-in Integrated Development Environment pre-compiler module, the main purpose of which to extend the basic features of the low-level language syntax to high level syntax.
Chapter 14 – Script Example.
Chapter 15 – FlexDC Script Keywords Commands Reference. This chapter lists macro related commands supported by the FlexDC.

Document Structure (continued)

Part IV – Nanomotion Shell Application
Chapter 16 – Introduction
Chapter 17 – Software Installation
Chapter 18 – The Nanomotion Shell Application GUI
Chapter 19 – "Srcedit" – the Macro File Editor Application
Part V– SCServer COM/DCOM Interface Library. This part describes the COM software interface module for the Nanomotion FlexDC servo controller.
Chapter 20 – Introduction
Chapter 21 – Getting Started
Chapter 22 – Object Parameters and Methods Syntax
Chapter 23 – Object Definitions
Part VI– Communication Library (Commdll.dll). The "Comdll.dll" library enables the user and interface to COM ports, using the RS232 communication.
Chapter 24 – Introduction
Chapter 25 – The Communication Library - COMDLL.DLL
Chapter 26 – Communication Error Codes
Chapter 27 – Code Example
Part VII – Glossary

1.3 FlexDC Installation and Operation Sequence



Part II – FlexDC Software and Commands Reference

2 Commands Syntax and Protocols

2.1 Introduction

This chapter focuses on the FlexDC communication syntax, including response to commands clauses and errors.

2.2 Supported Communication Protocols

The FlexDC currently supports two basic communication protocols and channels:

- ASCII based RS232
- Binary CAN bus.

Using separate hardware interface layers, the RS232 and CAN bus communication links (and their protocols) are completely independent from one another, and can be used simultaneously (excluding few special cases as described in section 2.2.1 below).

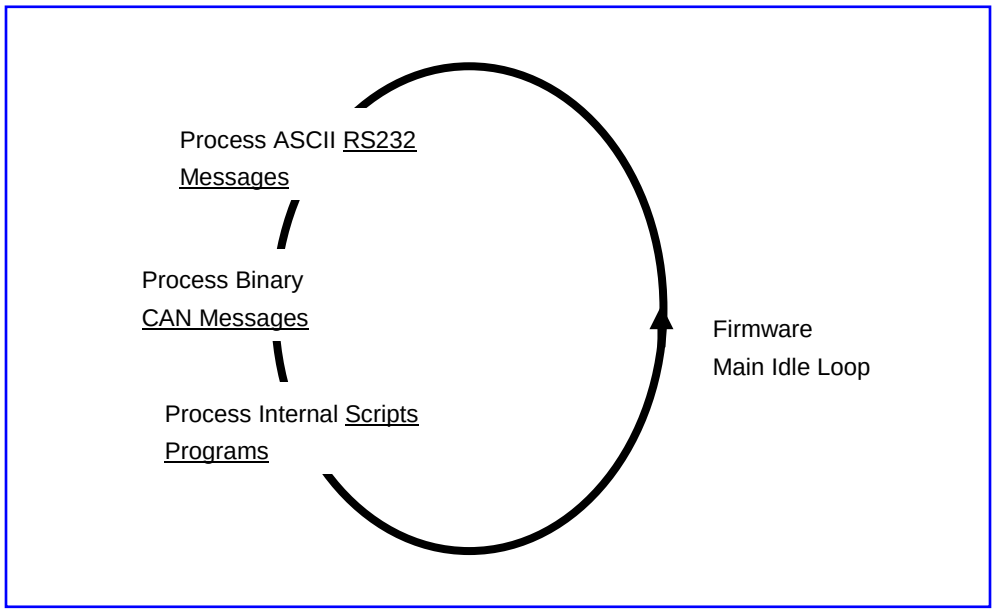


Figure 1: Communication Channels Handling within the Firmware Main Idle Loop

As shown in Figure 1, the servo controller firmware main loop is continuously monitoring both communication channels, handling incoming messages separate from one another. This is possible in the FlexDC firmware and syntax architecture as almost all keywords and commands are executed immediately without blocking any other processes. The complete "bits and bytes" comprehensive description of each one of the protocols is fully covered, further in this user manual.

2.2.1 Simultaneous Communication Channels Operation Support

As discussed above, both communications protocols can operate simultaneously without any interference. This is possible in the FlexDC architecture as almost all keywords and commands are executed immediately without blocking any other processes.

However, there are certain cases where a special operation in one channel can block the other. These cases are:

- When downloading new firmware in RS232 (supported ONLY in RS232), all other channels are immediately disabled.
- When downloading a new user program in one of the channels, the other channel is blocked for the same operation. Other communication with the second channel is fully functional.
- When uploading large arrays in one channel, other channels are blocked until the upload operation is completed.

2.3 FlexDC Communication Language Definitions

2.3.1 General

In the following sub-sections, the FlexDC basic communication language is defined.

It should be noted that the same "Language Syntax Rules" apply, regardless of the command source, which can be one of the following: RS232 communication, CAN Bus communication, possible future supported communication links, and the internal script program engine.

When a new command is received from either one of the channels described above, its source is recorded for later reference, and the command itself is passed to an

internal software module “The Command Interpreter”, which checks its syntax, and if a valid command is detected, executes the command.

2.3.2 Language Notations

The communication keywords are divided into two groups of keywords:

- Parameters Keywords.
- Command Keywords.

The execution time of a parameter keyword is minimal and usually negligible (few micro-seconds at most). The execution time of a command may be longer (for example: save parameters, or upload list data). Find the definitions of each keyword type group below.

2.3.2.1 Parameters Keywords

Parameters can always report their value (generally reflecting the value of an internal software or hardware register) and in most cases can be assigned with a value. There are some read only parameters that cannot be assigned with a new value. For example, the “AI” (Analog Input value) is a read only parameter.

There are some parameters that when assigned with a new value, can also modify the values of other parameters. For example, when modifying the “PS” (Current Encoder Position Value) of an axis, the “DP” (The current position command reference or Desired Position) is also modified to the same value to avoid positioning errors.

2.3.2.2 Command Keywords

Command keywords always initiate a process (start a motion, save parameters, begin internal script program execution, etc.). Commands do not report a specific register values, and in general, do not assign any specific register values, though they can internally modify the values of more then one register. For example, the “BR” (Begin Recording) command modifies the value of the “RR” (Recording Status) register. The “LD” (Load from Flash Memory) command modifies values of almost ALL registers.

Commands can receive a parameter (actually an argument) which affects the command process. For example, the command to execute a program (“QE”)

can receive a label string argument, indicating the name of the subroutine to execute (e.g. “XQE,#HOME”). Command’s parameter can be a string (see above), or a number. The command’s parameter is separated from the command itself using a comma “**,**” character.

2.3.3 Keywords Attributes and Restrictions

Each keyword has attributes defining it, and restrictions that must be satisfied in order to accept the command clause. The Command Interpreter module checks the restrictions before actually executing the command or making a parameter assignment. For parameters, the restrictions relate only to assignment, since reporting is always valid.

Restrictions, for both parameters and commands, may include one or more of the following list:

- **None:** No restriction is applicable.
- **Motor should be ON (0x00000001):** The requested command or parameter assignment needs an enabled motor. For example, the “BG” (begin motion) command must have its related motor enabled in order to be executed successfully.
- **Motor should be OFF (0x00000002):** The requested command or parameter assignment needs a disabled motor. For example, the “CG” (axis configuration) parameter can be assigned with a new value ONLY if its related motor is disabled. The assignment can not be executed if the motor is enabled.
- **Motion should be ON (0x00000004):** The requested command or parameter assignment can be executed only if a motion is currently being executed.

- **Motion should be OFF (0x00000008):** The requested command or parameter assignment can be executed only if there is no current motion. For example, the Motion Mode (“MM”) parameter can not be changed during motion.
- **Parameter is Read only (0x00000010):** A Read-Only parameter can only be inquired for its value. The user can not assign values for Read-Only parameters. For example, “DP” (the current reference Desired Position value) is a read only parameter, and can not be directly assigned a new value by the user.
- **Keyword Source MUST be an internal program (0x00100000):** The keyword can only be used from an internal script program. For example, the “RT” (return from subroutine) command can only be called from with in a program subroutine.
- **Keyword Source MUST be external Communication (0x00200000):** The keyword can only be used from an external communication link. For example, the “QD” (download a new program) command can only be called from an external communication link.
- **Keyword Source MUST be RS232 Communication (0x00400000):** The keyword can only be called from an RS232 link. For example downloading new Firmware is supported ONLY in RS232 mode.
- **Keyword Source MUST have all internal programs halted (0x10000000):** The keyword can only be executed when all internal user programs are halted. For example, the “LD” command (Load from Flash Memory), can be called only in that case.

Parameter values always have a minimum and maximum value for assignment clauses. Most parameters are saved to Flash Memory. Few are initialized to default non-active values at power ON, reset, or Load from Flash memory events.

2.4 Axes and Groups Identifiers

The FlexDC supports Group Definitions for Axes Identifiers. The FlexDC language syntax requires an axis identifier before any keyword:

- '**X**', '**Y**' : FlexDC Axis Prefixes (for physical axes). When a specific axis identifier is given, the command interpreter interprets the clause and acts upon the specific axis only.
- '**B**' for both 'X' and 'Y' axes. The FlexDC command interpreter supports a notation of the one Axes Group Identifier. The user can perform an action on more then one axis simultaneously, for example, reporting the position of all axes at once: the Axes Group "B". This group defines X-axis and Y-axis as sub-groups. For example, issuing the following assignment "BPS=0" sets the position of both axes (X and Y) to "0".

2.4.1 FlexDC Axes Attributes

In addition to the restrictions and attributes described above, each keyword in the servo controller has also an "Axes Relation Attribute Field".

The "Axes Relation Attribute Field" defines whether the keyword is axis related or not, and also defines to which of the axes the keyword can be related and to which not. Below is a list that describes the possible "Axes Relation Attributes" supported by the controller:

Keyword Axes Attributes	**Axes Supported by the Keyword**
None	Irrelevant to the axis. The keyword is NOT axis related.
CPA_KW_IS_AXIS_RELATED	Keyword is supported on axes X and Y (both axes)

2.5 RS232 Communication

This section defines the RS232 communication protocol and syntax, including response to commands clauses and errors. This section presents the general Language Syntax of the FlexDC controller's software as well. Note that the RS232 ASCII protocol, the CAN bus protocol is logically similar.

2.5.1 Hardware Interfaces

The FlexDC supports the following RS232 hardware Interface:

- RS232, 3 wires, no hardware handshaking.
- 8 bits, 1 start bit, 1 stop bits, no parity.
- Baud-rates of: 38,400 Baud; 115,200 Baud.

2.5.2 Connecting and Defining the RS232

Connect the FlexDC and define the RS232 connection, according to the detailed instructions, given in the "Quick Start" Chapter, in the "FlexDC User Manual".

2.5.3 Language Syntax – Host to FlexDC

2.5.3.1 Keywords

Each keyword consists of two upper case letters. Some of the parameters are defined as arrays. These parameters are always referred with their two letters keyword and with an index number within a square brackets, e.g. AR[2].

2.5.3.2 Clause Termination

Each command clause is terminated with a terminator character, which may be one of the following:

- "CR>" : Carriage Return,
- ";" : semicolon.

2.5.3.3 Axis Identification in Clause

Each command clause is preceded with an Axis Identification Letter, to identify the axis to which the command clause is addressed, see paragraph 2.4, Part II.

2.5.3.4 Axis Related Keywords and Clause Axis Identifiers

Some of the command clauses are not axis related (e.g.: "SV" for saving parameters to the Flash Memory or the "AR" for the Data Array), in these cases the axis identification letter is ignored, although it still MUST be included. Calling a keyword with no axis identifier pre-fix is an error.

2.5.3.5 Clause Handling

A command clause is handled only after the termination character has been received. Command clause characters are received (buffer) but are not handled until the current command handling is completed.

Each command clause includes only a single keyword, which may be a command or a parameter.

In the case of a command keyword, the command clause includes only the command keyword (preceded with the axis identification letter). In the case of a parameter keyword, the command clause may be a report or a set parameter clause.

A report parameter value command clause includes only the parameter keyword (with index in square brackets for arrays).

A set parameter value command clause includes the parameter keyword (with index in square brackets for arrays), “=” and the value. Parameter values can be in decimal form, long integers and in text format (ASCII printable characters).

Notes:

- *Blanks, tabs and new-line characters are received, echoed but ignored.*
- *Back-spaces are handled.*

Examples:

XSP <CR> Report parameter clauses
YSP ;
XAR[5]<CR>
YSP= 10000; Set parameter clauses
BAC = 1000000 <CR>
BAR[3]=345 ;
XBG <CR> Commands
BST ;

2.5.4 Language Syntax – FlexDC to Host

In case of a report parameter clause, the reported value is sent back to the host (decimal, long integer, text format).

Each character (including blanks, tabs, new-line and terminators) are echoed as is, unless otherwise selected by the user "EO" command (Echo ON/OFF).

In case of a report parameter clause, the reported value is sent back to the host (decimal, long integer, text format in RS232, and binary format in CAN bus).

2.5.4.1 Clause Prompts

After handling each command clause, a prompt is sent back to the host computer. The prompt is “>” in the case of a successful command clause execution or “?>” in the case of any error in the execution of the command clause (command was not executed).

In the latter case, a dedicated parameter "EC" (Error Code) stores the code of the last communication error. In cases when the last error was generated in a user script program, another dedicated parameter stores the last program Error Code. "QC" (Program Error Code). For a complete description of all currently supported error codes, refer to chapter 15, Part II, in this user manual.

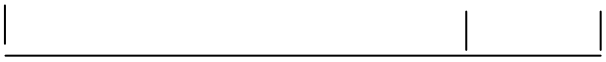
Notes:

- *An empty command clause is a legal “do nothing” command (can serve as a “ping” command).*
- *The prompt is sent only after the clause execution is completed.*

Examples:

- *The italics strings are the FlexDC responses to the Host computer.*
- *The blanks are only for the clarity of the example and the send/get timing.*
- *Setting "SP" (Speed of X axis) to 10,000:*

X S P = 1 0 0 0 0 ;
X S P = 1 0 0 0 0 ; >



Echo (only if EO=1) Response (always sent)

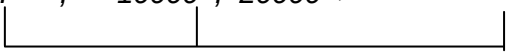
- *Setting "AC" (Acceleration of X and Y by default) to 10,000:*

B A C = 1 0 0 0 0 0 0 CR
B A C = 1 0 0 0 0 0 0 CR >

Echo (only if EO=1) Response (always sent)

- *Reporting the value of the SP (for X and Y by default):*

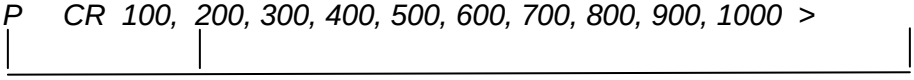
B S P ;
B S P ; 10000 , 20000 >



Echo (only if EO=1) Response (always sent)

- *Reporting the value of the SP (assuming A is configured to all axes):*

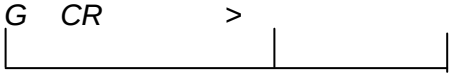
A S P CR
A S P CR 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 >



Echo (only if EO=1) Response (always sent)

- *Executing a Begin Motion Command for X and Y by default:*

B B G CR
B B G CR >



Echo (only if EO=1) Response (always sent)

- *Trying to assign an out of range value to YAC (error prompt is sent, and EC is updated accordingly):*

```
Y  A  C  =  -  1  0  0  0  CR
  Y  A  C  =  -  1  0  0  0  CR  ?  >
```

Echo (only if EO=1) Response (always sent)

- *Executing a Script function named "HOME_X" in Program #1 (X):*

```
X  Q  E  ,  # HOME_X  CR
  X  Q  E  ,  # HOME_X  CR  >
```

Echo (only if EO=1) Response (always sent)

2.6 Ethernet/LAN Communication

This chapter describes how to connect the FlexDC to a PC via Ethernet and establish the communication.

2.6.1 FlexDC Network Topologies

In order to obtain best communication performance with the controller, it is recommended to connect the controller to a dedicated scheduled network, apart from a global organization network. Supporting numerous Ethernet cards is standard nowadays, and can be easily configured. This network may consist of a single computer and a controller (Single Computer/Controller network); or of numerous computers with numerous controllers, all connected via a hub / switch (Multi computer/controller network).

In Figure 2, the FlexDC controllers are part of a separate network, and are connected to a switch using **standard, non-crossed** Ethernet cable (CAT 5E). Smart switches have the ability to detect crossed cables, and act correctly.

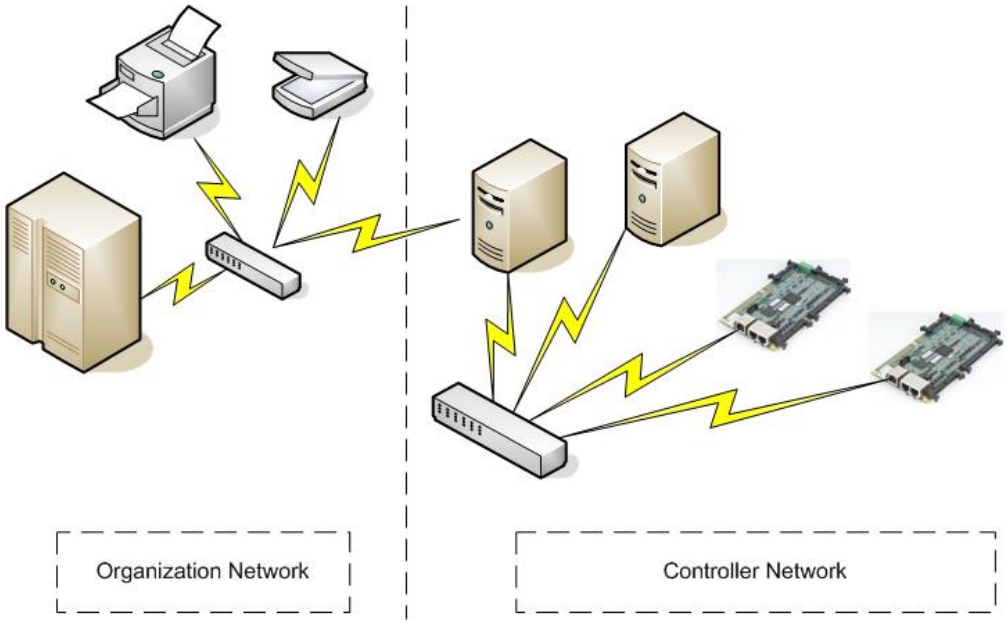


Figure 2: Milti-Computer / Controller Network

In Figure 3 the FlexDC controller is directly connected to the PC. In this case, the Ethernet cable, **must be a crossed** cable (CAT 5E).

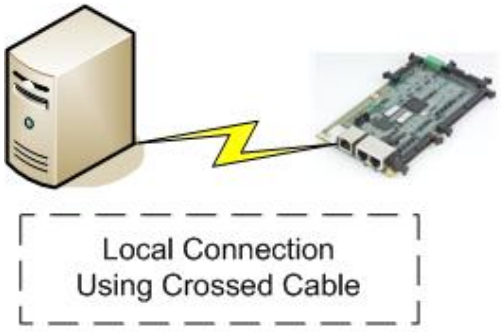


Figure 3: Single Computer / Controller Network

In either way, the IP addresses of the PC and the controllers must be statically defined, as there is no DHCP server that dynamically defines these addresses (see section 2.6.2, Part II).

2.6.2 Connecting and Defining the Ethernet/LAN

2.6.2.1 Connecting the LAN

Connect the RS232 and the LAN cables between the controller and the PC (refer to the instructions given in the "Quick Start" chapter in the "FlexDC User Manual" for the RS232 connection and Nanomotion Shell Application installation), refer to Figure 4.

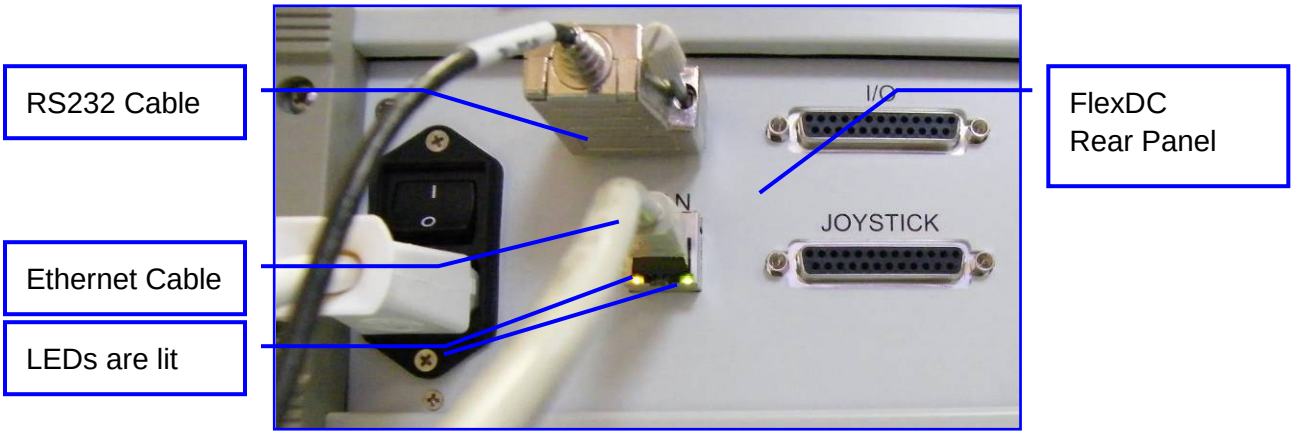


Figure 4: RS232 and LAN Connections

2.6.2.2 Setting Controller's IP Address

The FlexDC controller's IP address must be set by the user as part of controller's connection configuration setup. The controller cannot obtain an address automatically as a dynamic address; therefore the user must set the static IP address.

To set the controller's IP address, follow the next steps:

1. Open the Nanomotion Shell Application (refer to the "Quick Start" chapter in the "FlexDC User Manual", for Nanomotion Shell Application installation instructions).
2. On the Nanomotion Shell Application initial main screen select the LAN communication in the connection combo-box, see Figure 5.

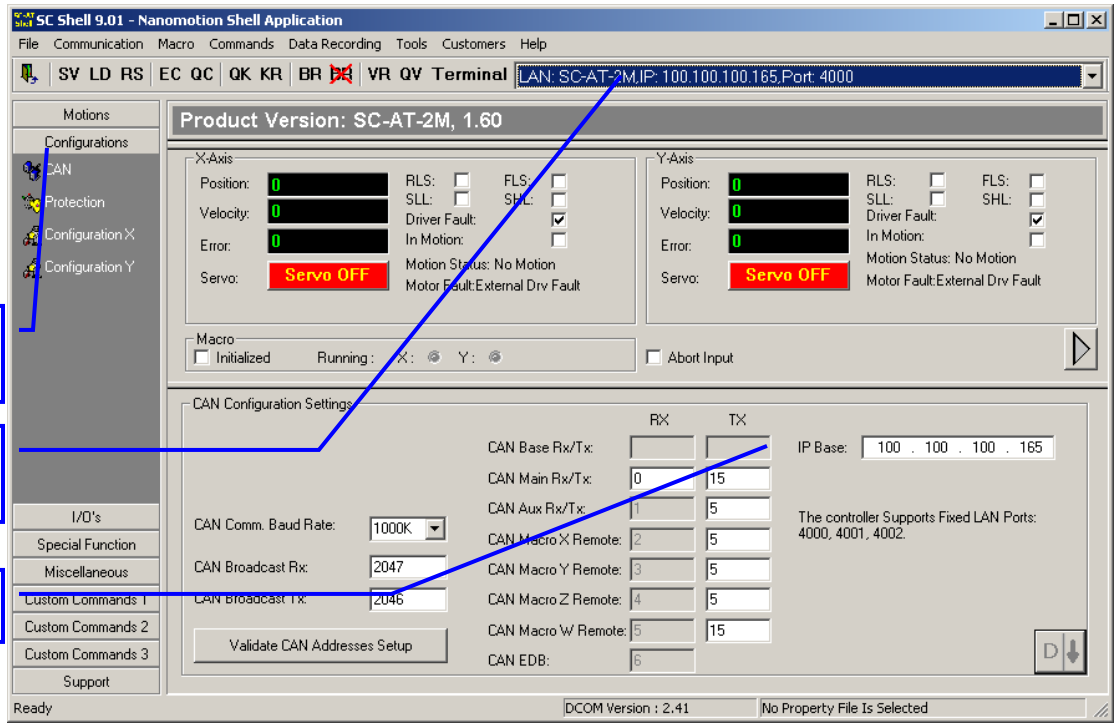
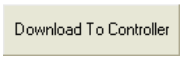


Figure 5: Nanomotion Shell Application – Configurations Menu

3. After the communication is established, click the "Configurations" menu on the left pane of the main screen.
4. Type the IP address in the "IP Base" window, see Figure 5.
5. Press the  button, and then choose  to download the data to the controller.
6. Press the  (Save) button on the main screen toolbar to save the parameters to the Flash Memory.
7. Press the  (Reset) button on the main screen toolbar to reset the controller.
8. Turn OFF and then turn ON the controller again, so the controller connects to the set IP address.
9. In order to validate the, previously set, on the "Configurations" menu (on the left pane), click the “CAN” icon.

10. The user can define additional specific options for the LAN connection: on the "Communication" menu click "Setup Communication Menu" (similar to RS232 connection, described in the "Quick Start" chapter in the "FlexDC User Manual").

- In the "Communication Settings" dialog box (see Figure 6) choose the "Controller Name".
- Type the IP address (set by the network administrator) in the "IP" window (see Figure 6).
- Define one of the following communication ports in "Ports" window: 4000, 4001, 4002, (see Figure 6).

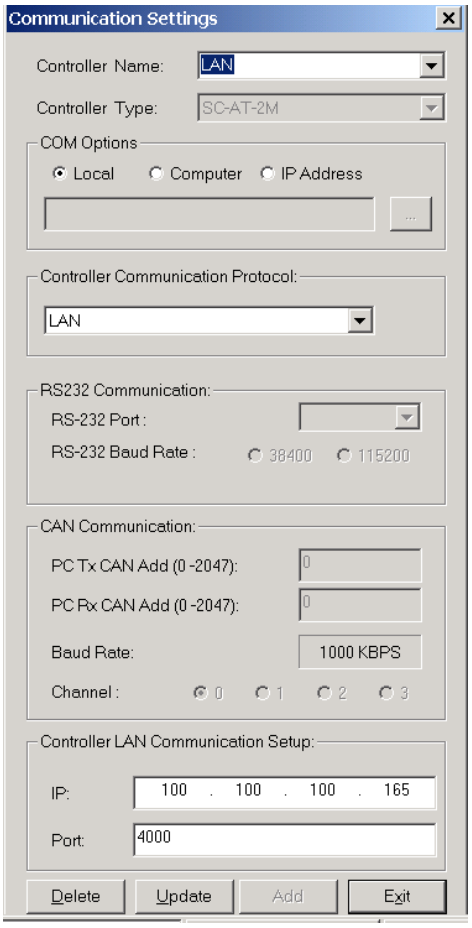


Figure 6: Communication Settings Dialog Box

11. Check the communication via the Nanomotion Shell Application connection combo-box.

2.6.3 TCP/IP Protocol (Ethernet / LAN)

In general, the raw data within the TCP/IP protocol is similar to the FlexDC RS232 protocol.

The user should comply with the following guidelines:

- All messages from the host to the controller end with <CR>. After the <CR>, a 1 byte message numerator appears.
- Blanks, tabs, back-spaces and new-line characters should not be sent.
- In case of a report parameter clause, the reported value is sent back to the host in decimal, long integer, text format.
- As opposed to the RS232 protocol, in the Ethernet protocol has no echo option. The clause received by the controller is sent back to the host.
- After handling a command or assignment clause, a prompt is sent back to the host computer. The prompt is “>” in the case of a successful command clause execution or “?>” in the case of any error in the execution of the command clause (command is not executed). In this case, a dedicated parameter "EC" (Error Code) stores the code of the last communication error.
- All messages include an ACK or NACK. (>, ?> or *value*>). After the ‘>’ sign, the identical numerator that was sent by the host, is sent back by the controller.
- Standard Ethernet protocol supports chaining numerous clauses with the ‘;’ character (semicolon). In this case, only one numerator be sent, after the CR. The controller's response reflects the order of the clauses sent, in a single message, divided by ‘;’. In a case where an axis grouping was sent as part of the clause, the replies to these clauses are be separated by the ‘,’ character (comma). A CR must appear. Chaining message length should be limited to 15 messages.

Examples:

Italics strings are the controller's responses to the host computer.

The blanks are only for the clarity of the example and the send/get timing.

X S P CR *8bitASCIINumerator*
10000 *> 8bitASCIINumerator*

Response (always sent)

X S P =90000 CR *8bitASCIINumerator*
> 8bitASCIINumerator

Response (always sent)

X A C = 200 CR *8bitASCIINumerator*
? *> 8bitASCIINumerator*

Response (always sent)

X S P =90000;YSP;XAC=200;BRP CR *8bitASCIINumerator*
>;50000>;?>;2000,3000>*8bitASCIINumerator*

Response (always sent)

2.6.4 Using DCOM Software Library for Ethernet Communication

The FlexDC is provided with a free DCOM (Distributed Component Object Model) software library, enabling easy and immediate interface with the customer's application program, supporting Visual C++, Visual Basic, LabView and others.

The COM (Component Object Model) server interface supports Ethernet options. Refer to Part V for the COM / DCOM Interface Library for further information.

The only difference in setting the communication type is in the OpenDeviceEx function. Once this function is called, (when the connection is defined), all access to the controller is over the Ethernet link, and is transparent to the user/programmer. The user needs no prior knowledge in TCP/IP in order to communicate with the controller.

The following is an example for calling OpenDeviceEx function:

```
OpenDeviceEx(
DeviceID                          // Device id
eCrsControllerType_SC_NT_2M       // controller type (FlexDC)
eCrsCommunicationType_ETHERNET    // Ethernet communication
0                                 // Reserved
LanPort                           // Flex supports one of the three
                                  ports(any or all): 4000/4001/4002
0                                 // Reserved
IPAddress                         // See below for format
0                                 // Reserved
0                                 // Reserved
0                                 // Reserved
0                                 // Reserved
0                                 // Reserved
```

The TCP/IP Port can be one of the following: 4000,4001,4002.

The TCP/IP address format is according to the following:

Lets assume we have the following IP address:

IPAddress | Byte-1 | Byte-2 | Byte-3 | Byte-4 |

The conversion is performed as follows:

IPAddress = (Byte-1 << 24) + (Byte-2 << 16) + (Byte-3 << 8) + (Byte-1)

In addition to the standard RS232 and CAN Bus communication library interfaces, the FlexDC also recognizes the Microsoft standard DCOM Server Interface, which provides easier access to professional or novice programmers to communicate with the controller. Only high-level keywords knowledge is needed.

2.7 CAN Communication

2.7.1 General

One of the main objectives of the CAN bus interface in the FlexDC is to allow for an additional communication interface which is much faster than the RS232 and is easy to implement and access.

CAN communication uses binary language syntax. In general, there are many common features shared by the RS232 and CAN bus command syntax. However, there are some important differences.

The exceptions are as follows:

- Most command clauses (all non-special commands) are limited to 8 characters, i.e. limited to one CAN bus message. Note that not all messages include the full 8 bytes message length.
- Special commands may include more than one full CAN message. See section 2.7.3, Part II, for more details.
- Since command clauses may be longer than 8 characters, a different (binary) format is to be used to encapsulate the RS232 command clause into 8 characters (except for the special commands, as noted before).
- Each response to a command clause is also limited to 8 characters (unless otherwise noted).
- CAN communication ignores the echo mode and does not send an echo. Prompt is still sent as usual (except for the special modes).
- Since the CAN hardware controller also indicates the message size, no termination is used in the communication protocol.

2.7.2 Language Syntax - Host to FlexDC

Normally, CAN communication message has basic 8 bytes structure, (messages shorter than 8 bytes are also valid).

The first byte in each message is a fixed structure Pre-Fix Byte. The next section describes the Pre-Fix Byte Structure.

2.7.2.1 The CAN Bus Message Pre-Fix Definition

Each clause sent from the Host to the controller must include a first byte (Pre-Fix) which describes the clause attributes. This pre-fix byte must include the following information:

2.7.2.1.1 CAN Bus Message Pre-Fix Definition

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Reserved	Special Command	Commands Parameter	Normal command (not an array command)	Command has data	Axis ID	Axis ID

Table 1: CAN Bus Pre-Fix Byte Format

- Bits 1, 0

Much like the RS232 communication interface, the CAN bus pre-fix byte includes a binary bit field (2 bits) that indicates the axis identifier. Please note that logically, the same axes identifiers as in RS232 are used as listed in Table 2:

Bit #1	Bit #0	Axis Identifier	Mask (Hex 4 Bits)
0	0	PING Message	0x0
0	1	X	0x1
1	0	Y	0x2
1	1	B	0x3

Table 2: Pre-Fix Axes identifiers

- Bit 2

This bit indicates whether the clause will include data or not. (‘1’ if clause will include data, ‘0’ if not). The data may be either Assignment data or Commands Parameter data.

- Bit 3

This bit indicates if a parameter clause is a normal parameter (i.e. not an array parameter, which means that there is no index required). If this bit is set to ‘1’, then this is a normal command (not an array). If the bit is ‘0’ then the parameter is an array parameter, i.e. an array code and index is expected. See section 2.7.2.3 for "Non-Normal clauses – (Array Clauses)", Part II.

- Bit 4

This bit indicates whether the data element in a message is a ‘Command’s Parameter’ or a ‘Parameter Assignment’.

When the bit is ‘0’, then a normal parameter assignment (i.e. with the ‘=’ sign is requested, like RS232 XPS=1234 parameter assignment for example). This bit should be ‘0’ for all standard parameter assignments.

When the bit is ‘1’, then the data in the message is referred to as a ‘Command’s Parameter’ (given with a ‘,’), when a command is expected to receive a parameter. For example, the command ‘XQE’ (execute a program) may be issued with no parameter at all, and then the program will start running from the current program pointer. If the user wants to start executing the program from a given label (or pointer), the label (or pointer) should be given as a ‘Command’s Parameter’. In RS232 the syntax will be: XQE,#LABEL1 (or XQE,1234 for a pointer). In CAN, the ‘Command’s Parameter’ bit should be set to indicate this case.

- Bit 5

This bit indicates that a 'Special Command' is issued. Currently only a label (#) command is supported as a special command. When this bit is set to '1' the controller reacts according to the special case for each command. See section 2.7.3 for "Special Commands", Part II.

2.7.2.2 Normal Clauses

Assuming the Normal bit (bit3) is '1', the following command structure is expected, see Table 3:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Pre-fix	KeyW.	KeyW.	Data(3) (optional)	Data(2) (optional)	Data(1) (optional)	Data(0) (optional)	

Table 3: Normal Clause CAN Bus Message Format

2.7.2.2.1 Normal Clauses Bytes Description

Note:

- *Back-spaces, blanks, tabs and new-line characters are not valid under CAN.*

- Byte 1

A pre-fix, as described above, including clause attributes (AxisID, data). See section 2.7.2.1 above.

- Bytes 2-3

Two bytes representing clause ASCII Keyword (same as RS232 communication keywords). All the controller keywords as described in chapter 7, Part II, are valid here, except those representing Arrays (not normal clauses).

- Bytes 4-7 (optional)

These 4 bytes are optional, and includes a binary data element (signed long representation). Note that this parameter is optional, and is regarded by the controller only if the Pre-Fix Data bit (bit 2) is '1', i.e. clause includes data.

- Byte 8:

Not used. Must be empty for future compatibility.

2.7.2.2.2 Normal Clauses Examples

The report X SP clause will be represented in RS232 by:

XSP <CR>RS232 Set parameter clauses

and in CAN by:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Pre-fix	KeyW.	KeyW.	Data(3) (optional)	Data(2) (optional)	Data(1) (optional)	Data(0) (optional)	
0x09	'S'	'P'					

Message length is 3 bytes.

The set Y AC 10000 clause will be represented in RS232 by:

YAC= 10000 <CR> RS232 Set parameter clauses

and in CAN by:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Pre-fix	KeyW.	KeyW.	Data(3) (optional)	Data(2) (optional)	Data(1) (optional)	Data(0) (optional)	
0x0e	'A'	'C'	0x00	0x00	0x27	0x10	

Message length is 7 bytes.

2.7.2.2.3 Ping Request

A Ping command is supported under CAN to allow the RS232 <CR> like command.

A message with length 1 byte that includes only the Normal bit set to '1' will be responded with an OK prompt.

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Pre-fix							
0x08							

This may be used if the controller is responding to communication messages at all.

Message length is 1 byte.

2.7.2.3 Non-Normal clauses – (Array Clauses)

Assuming the Normal bit (bit3) is '0', the following command structure is expected, see Table 4:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Pre-fix	Array Code	Array Index-1	Array Index-0	Data(3) (optional)	Data(2) (optional)	Data(1) (optional)	Data(0) (optional)

Table 4: Non-Normal Array Clause CAN Bus Message Format

2.7.2.3.1 Non-Normal Clauses (Array Clauses) Bytes Description

Notes:

- *Blanks, tabs and new-line characters are not valid under CAN.*
- *Back-spaces are not valid under CAN.*

  - Byte 1

  A pre-fix, as described above, including clause attributes (X/Y/B, data ?).

  - Byte 2

  A byte represents the Array Code. Each array (in addition to its name) has a code. The code must be included in this byte (in binary format).

  - Bytes 3-4

  Two bytes representing a binary array index, in unsigned short format (replacing the RS232 ASCII [Array Index]). The valid range for the array index is the same as in RS232 communication, and described in chapter 7, Part II.

  - Bytes 5-8 (optional)

  These 4 bytes are optional, and includes a binary data element (signed long representation). Note that this parameter is optional, and is regarded by the controller only if the Pre-Fix Data bit (bit 2) is '1', i.e. clause includes data.

2.7.2.3.2 Non-Normal Clauses (Array Clauses) Examples

The report X AR[5] clause is represented in RS232 by:

XAR[5]<CR> RS232 Report Array member clauses

and in CAN by:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Pre-fix	Array Code	Array Index-1	Array Index-0	Data(3) (optional)	Data(2) (optional)	Data(1) (optional)	Data(0) (optional)
0x01	0x00	0x00	0x05				

Message length is 4 bytes.

The set B AR[1000] = 722 clause will be represented in RS232 by:

BAR[1000]=722<CR> RS232 Set Array member clauses

and in CAN by:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Pre-fix	Array Code	Array Index-1	Array Index-0	Data(3) (optional)	Data(2) (optional)	Data(1) (optional)	Data(0) (optional)
0x07	0x00	0x03	0xe8	0x00	0x00	0x02	0xd2

Message length is 8 bytes.

The report X PA[1] clause will be represented in RS232 by:

XPA[1]<CR> RS232 Report Array member clauses

And in CAN by:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Pre-fix	Array Code	Array Index-1	Array Index-0	Data(3) (optional)	Data(2) (optional)	Data(1) (optional)	Data(0) (optional)
0x01	0x05	0x00	0x01				

Message length is 4 bytes.

2.7.2.4 CAN Bus Array codes Description

FlexDC supports the following array codes:

Array	Array code
AR	0
Not used	1
IA	2
TD	3
Not used	4
PA	5
ZI	6
Not used	7
Not used	8
CA	9
DA	10
PG	11
Not used	12
Not used	13
Not used	14
FF	15
KD	16
KI	17
KP	18
RG	19
Not used	20
Not used	21
Not used	22
RV	23
AI	24
FR	25

Refer to chapter 7, Part II, for more information regarding the arrays.

2.7.3 Special Commands

Currently the only special command case supported is the Label (#) case. The label is required to execute a program from a specified location.

For example, the command ‘XQE’ (execute a program) may be issued with no parameter at all, and the program starts running from the current program pointer. If the user wants to start executing the program from a given label, the label should be given as a ‘Command’s Parameter’, and the command should be signaled as a special command.

Since the FlexDC supports up to 6 label characters by CAN, (otherwise it supports up to 12 characters), and the command itself includes 2 characters, a special format is defined.

In such cases, the command is split into 2 messages. The first message includes the label data only (with the pre-fix byte of course), and then the command itself (with no data) is sent. The ‘Special Command’ bit is used in these cases to indicate that the command is split into 2 messages.

For example, the following RS232 syntax: ZQE,#LABEL1 (meaning start executing the Z program from label #LABEL1) is issued in CAN as 2 consecutive messages:

The first message is as follows:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Pre-fix	Special Code	Label (byte 1)	Label (byte 1)	Label (byte 1)	Label (byte 1)	Label (byte 1)	Label (byte 1)
0x20	0x01	‘L’	‘A’	‘B’	‘E’	‘L’	‘ ‘

Message length should always be 8 bytes (labels are left justified padded with blanks to the right). Currently, the only special code supported is 0x01 (to indicate the label ‘#’ sign). The prefix includes the ‘Special Command’ bit only as ‘1’.

See the next message:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Pre-fix	KeyW.	KeyW.	Data1	Data2	Data3	Data4	
0x7d	‘Q’	‘E’	0	0	0	0	

Message length should be 7 bytes in this case. The prefix includes the ‘Z’ bit (refer above to axis prefixes), the ‘NORMAL’ bit (0x08), the ‘COMMANDS PARAMETER’ bit (0x10) and the ‘SPECIAL COMMAND’ bit (0x20). The data bit should be set, and data should be zero. The actual data is taken from the previous command. Note that although in this case both the ‘COMMANDS PARAMETER’ bit (0x10) and the ‘SPECIAL COMMAND’ bit (0x20) are set, in general this is dependent on the command used.

2.7.4 Language Syntax– FlexDC to Host

Clauses sent from the controller to the host may have one of the following structures:

2.7.4.1 Prompt OK

An OK response to a command or a set parameter (normal prompt as in RS232):

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
‘>’							

Message length is 1 byte only.

2.7.4.2 Prompt Not OK

An error response to a command or a set parameter (normal prompt as in RS232):

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
‘?’	‘>’						

Message length is 2 bytes. In order to retrieve the error code, an EC command can be issued to the controller.

2.7.4.3 Prompt Including a Single Axis Report Response

If a report command was issued for X or Y, the controller is responding with the requested data, followed by the prompt (similar to RS232):

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Data(3)	Data(2)	Data(1)	Data(0)				

Message length is 4 bytes. Data is represented binary, signed long format. Note that no '>' is returned in this case.

2.7.4.4 Prompt Including a Report Response for Two Axes

If a report command was issued for both axes (BAC, AAC) for example), the controller responds with the requested data, with no prompt.

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Data(3) Axis1	Data(2) Axis1	Data(1) Axis1	Data(0) Axis1	Data(3) Axsi2	Data(2) Axis2	Data(1) Axis2	Data(0) Axis2

Message length is 8 bytes. Data is represented in binary, signed long format.

Note:

- *The CAN report can return a response to a maximum of two axes.*

2.7.4.5 Controller Initiated CAN Messages

FlexDC's macro program is able to send initiated messages from the controller to a host.

The command to initiate CAN messages from the controller script program to a host PC is "ZM". For more information regarding the "ZM" command refer to chapter 7, Part II.

The controller can send the following formats:

- One report variable – the format of the data in the CAN message is identical to section 2.7.4.3 above.
- Two report variables – the format of the data in the CAN message is identical to section 2.7.4.4 above.

A string, up to a length of 8 bytes – the host can identify the length of the CAN message sent. The message bytes are in the order of the string sent from the controller.

2.7.5 CAN - Enhanced Download Buffer Mode (EDB)

This mode enables a host computer to download large quantities of CAN data to the AR array.

In this mode, the controller continuously listens to a dedicated CAN address, and monitors all messages received in it. According to a set of parameters, the controller then stores the incoming data in the relevant buffers, and auto-increments the store location for both buffers separately.

The new message format (received by the controller) is:

CAN Byte #	0	1	2	3	4	5	6	7
Data Format	D1-LSB	---	---	D1-MSB	D2-LSB	---	---	D2-MSB
Data	Long Data #1				Long Data #2			

While using this mode, keep with the following guidelines:

- If the received message length is less than 4, the message is ignored.
- If the received message length is equal to 4, only Long Data #1 is assumed valid, and only a single buffer is updated (Buffer #1).
- If the received message length is less than 8, the message is ignored.
- If the received message length is equal to 8, both Long Data #1 and Long Data #2 are assumed to be valid, and both buffers (#1 and #2) are updated correspondingly.

2.7.5.1 Message Format

Message Bytes order is in little endian format. This is NOT compliant with the Download Buffer Message format which is inverted (MSB first).

2.7.5.2 EDB Data Validity Check

No validity check to the data value itself is performed. In any case a buffer is updated with new data; its storing location is defined in a special new dedicated parameter, and is auto-incremented automatically.

2.7.5.3 EDB Buffers

The ***Buffer*** is a valid FlexDC array member (AR[ ], for example), but the EDB mode is not limited to AR only. In the EDB mode the user can select to download to any valid FlexDC array element as explained below.

Furthermore, the user can select to download Data Long #1 to one array, and Data Long #2 to another array.

The following FlexDC parameters in Table 5 are used for the implementation of the “Enhanced Download Buffer Mode” (EDB):

XZI[4]	Buffer #1 Array Code This parameter defines the code for the requested Array to be updated by the parameter “Long Data #1” in the incoming EDB message. **Not supported in current FlexDC firmware.**
YZI[4]	Buffer #1 Axis Code This parameter defines the axis to be updated in axis related arrays. 1 is X, 2 is Y, etc. If a non-axis related array is assigned, this parameter MUST BE “1”. Note that the controller does not validate this parameter according to the Array type code. See section 2.7.2.4.
ZZI[4]	Buffer #1 Current Index This parameter is used as the current index to which the new data is stored (in the selected Array and Axis). The controller ONLY checks that the current index location is valid for the specific selected array (not for a specific axis!). If the store index is valid, the index is being auto-incremented (after the store) with the Auto-Increment parameter value (see below).
WZI[4]	Buffer #1 Increment Value This parameter is used as the Auto Increment value in case a valid store is executed. This number value is not validated.
EZI[4]	Buffer #2 Array Code This parameter defines the code for the requested Array to be updated by the parameter “Long Data #2” in the incoming EDB message. Refer above to Array codes. **Not supported in current FlexDC firmware.**
FZI[4]	Buffer #2 Axis Code Same as for Buffer #1, to be applied to Buffer #2.
GZI[4]	Buffer #2 Current Index Same as for Buffer #1, to be applied to Buffer #2.
HZI[4]	Buffer #2 Increment Value Same as for Buffer #1, to be applied to Buffer #2.

Table 5: EDB Buffers

2.7.5.4 EDB Mode Limitations

- The EDB mode is supported in CAN bus only and not supported in RS232.
- Unlike the standard DB (Download Buffer) FlexDC command, where the controller enters a special communication mode, the EDB mode works continuously and in parallel to normal RS232 and CAN messages (if the EDB is enabled, of course). This means that when data is downloaded through the EDB message, the user can communicate with the controller normally (with exceptions like download macro, upload data recording buffers, etc.).
- When a prompt is requested, the user should always wait for the reply before sending any other data to the controller (like any other normal clause).
- In the EDB mode, the controller only validates that the array code and the current store index (for each buffer) are legal. The controller neither checks that the axis number matches the array type, nor validates the data range. It is the user's responsibility to guarantee the correct initial configuration.

2.7.5.5 EDB Mode Example

In order to use the EDB mode, the host software should initialize the following parameters. This assumes that the user downloads to the buffer #2, i.e. the Axis1 data is downloaded to AR[301÷600], and the Axis2 data is loaded to AR[1301÷1600].

2.7.5.6 Command Sequence

YZI[4]	= 1	// Enable EDB the Mode.
YZI[4]	= 256	// Enable EDB Prompt Mode.
YZI[5]	= 5	// Set The EDB Rx CAN Address.
XZI[6]	= 0	// Set first data array to AR[] Array
XZI[7]	= 1	// Axis is 1 for Non Axis Related Array.
XZI[8]	= 301	// For Start Index AR[301]
XZI[9]	= 1	// Inc. Index #1 by “1” each message.
YZI[6]	= 0	// Set second data array to AR[] Array
YZI[7]	= 1	// Axis is 1 for Non Axis Related Array.
YZI[8]	= 1301	// For Start Index AR[1]
YZI[9]	= 1	// Inc. index #2 by “1” each message.

Once the above definitions are given, the host should start downloading the EDB messages in the format described above (8 bytes per message, where the lower 4 bytes are the data to the main scan axis buffer, and the upper 4 bytes are for the orthogonal axis). For each new slice, the host needs to modify only the Current Index (next Start Point).

During operation, the user can inquire the value of the “**EDB Mode Error Status Report”** to check for EDB message errors.

3 Motion Modes

This chapter describes the various motion modes supported by the FlexDC. Motion mode defines a type of motion. The exact motion, for each Motion Mode, is defined by a set of related parameters, such as speed (SP), acceleration (AC) and many other parameters.

While most of these parameters can be modified on-the-fly during active motion (practically affecting the motion profile), the motion mode itself can not be modified during active motion.

The motion mode for the FlexDC is defined by a combination of two parameters: MM (Motion Mode) and SM (Special Motion Mode).

Most standard motion modes are defined by the value of MM, with SM=0. Some special motion modes use both MM (to define the basic motion mode) and SM (to define a special variation of it).

The following sections describe each motion mode.

Notes:

- *The communication clauses given in the following sections for how to start/stop and monitor each motion mode are just examples. A specific application can use any desired value for the related parameters (such as acceleration and speed).*
- *The values of most parameters do not need to be sent again before each motion. The FlexDC uses the current value of each parameter when a new motion is commanded.*
- *Sometimes we use a semicolon ';' mark between commands. This is simply to save space. The user can use the ';' in all commands, or use none.*

3.1 Point to Point – PTP (MM=0, SM=0)

In this mode the controller calculates a standard smoothed trapezoidal profile from the current position to a user specified target position, using a user specified acceleration and speed.

The profile is called trapezoidal since the velocity command has a trapezoidal (or triangular for short distances) shape. The user can select to smooth the profile in order “round” the sharp trapezoidal (or triangular) corners. If smoothing is used, then the actual jerks are limited (no zero time acceleration change). Without smoothing, the jerks are infinite (acceleration is changed at “0” time).

The target position can be specified relatively to the current desired position, using the RP (Relative Position) parameter. It can be also specified as an absolute position, using the AP (Absolute Position) parameter.

It is important to note that PTP motion is always executed toward the value of the AP parameter. However, sending an RP=<value> clause is internally interpreted as:

AP=DP+<value>,

where DP is the current desired position (normally equal to the current actual position).

As a result, the AP is indeed modified when a new value is assigned to RP, and any following PTP motion toward AP actually moves to the desired relative position.

The only disadvantage of this method is that, for repeated relative motions, RP should be sent again before each motion.

The FlexDC supports separate AC (Acceleration) and DC (Deceleration) values in all profile based motion types. Furthermore, a new DL (Deceleration on Limit) parameter is supported in order to define a special Deceleration values when Limits are hit (works both for software and hardware limits).

3.1.1 Starting a PTP Motion

Communication Clauses	Description
MO=1	Enabling the servo loop, motor ON
MM=0;SM=0	Setting PTP motion mode
AC=500000	Assigning a value for the acceleration, [counts/sec^2]
DC=500000	Assigning a value for the deceleration, [counts/sec^2]
DL=1000000	Assigning a value for the Limit DC, [counts/sec^2]
WW=0	Defines no smoothing.
SP=50000	Assigning a value for the speed, [counts/sec]
AP=100000	Assigning an absolute target position, [counts]
RP=30000	**or**, assigning a relative value for the target position
BG	Begin the motion

3.1.2 Monitoring Motions

During and after active motion, the motion status can be continuously monitored using the following parameters. Note, that these parameters reflect the internal controller status regardless of the motion mode, and are relevant in all motion modes described below in this chapter.

The user can of course choose to record any of these variables (and many others) using the internal Data Recording capability.

Communication Clauses	Description
PS	Reports the current actual motor position, [counts].
VL	Reports the current actual motor speed, [counts/sec].
DP	Reports the current desired position, [counts].
PE	Reports the current Position Error (DP-PS), [counts]
MO	Reports the current motor status. Should be normally "1" for motor ON. "0" (OFF) only in case of fault during the motion.

Communication Clauses	Description
MF	A code describing why the motor was lastly disabled: MF=0: Motor was not disabled. MF=1: Driver's fault (Fault Input). MF=2: Abort input (emergency stop). MF=3: High Position Error (|PE| > ER). MF=4: Motor Stuck Condition. MF=65: Encoder Quad Error. MF=129: Encoder Dis-Connected Error.
MS	A bitwise code describing the current motion status: Bit 0: In motion. Bit 1: In stop. Bit 2: In acceleration. Bit 3: In deceleration. Bit 4: Waiting for input to start motion. Bit 5: In PTP stop (decelerating to target). Bit 6: Waiting for end of WT period.
SR	A bitwise code describing some controller statuses. Currently only Bit #5 (zero based) is reported. Other bits may be used in the future and should not be assumed to have any pre-defined value. Bit 5: In target[1].
EM	A code describing the cause for last end-of-motion: EM=0: Motion is still active. EM=1: Normal end-of-motion. EM=2: Forward limit switch (FLS). EM=3: Reverse limit switch (RLS). EM=4: High software limit (PS > HL). EM=5: Low software limit (PS < LL). EM=6: Motor was disabled (check MF). EM=7: User command (ST or AB). EM=8: Motor OFF by user (MO=0).

[1] This bit indicates that the motion profile has been finished and that the absolute position error (|PE|) is smaller than the target radius (TR) for at least target time (TT) consecutive samples (each 61 [μs]).

3.1.3 Stopping a Motion

A PTP motion is automatically finished when the desired position (the motion profile, not the actual motor position) reaches the desired target position. At this time the Motion Status (MS) is read as 0 and the controller is ready for a new motion or a new motion mode.

The EM (End Motion) parameter is set to 1, indicating normal end-of-motion.

A PTP motion can be also stopped by the following communication clauses:

Communication Clauses	Description
AB	Aborts the motion immediately (DP remains as its last value).
ST	Stops the motion with deceleration to zero speed.
MO=0	Disables the motor, effectively stopping any motion.

Of course, any software or hardware fault, limitation, or protection also immediately aborts or stops the motion (depending on the fault or limitation type). The last motion end reason can be monitored with the EM parameter.

3.1.4 On The Fly Parameters Change

The following parameters can be modified on-the-fly during active PTP motion:

Communication Clauses	Description
SP	Starts an acceleration or deceleration toward the new SP value.
AC,DC,DL	Defined new Accelerations and Decelerations for the current motion.
RP	Changes motion (including direction) to move toward the new AP (AP=DP+RP) value. RP can be modified even during deceleration to the previous target position and can be modified to any value, independent of the current position.
AP	Changes motion (including direction) to move toward the new AP. AP can be modified even during deceleration to the previous target position and can be modified to any value, independent of the current position.

Note that AP (or RP) change during motion may cause the motor to change its motion direction. This happens if a new AP value is given to a point that was already passed by the system.

3.2 Repetitive Point to Point – Rep PTP (MM=0, SM=1)

This mode is very similar to the standard PTP motion mode, as described above.

However, repetitive motion mode supports motions back and forth between two positions. Each motion is a standard PTP motion (uses SP, AC, DC etc. as described above) but the controller automatically generates the sequence of motions without the need to re-sending the BG command.

This mode is excellent for tuning the PIV filter. The motor is commanded to perform infinite motions back and forth, while the PIV parameters are modified on-the-fly to examine their effect on the motion performance (optionally using the Data Recording feature).

Two additional keywords are used for the Repetitive PTP mode:

- WT: Wait Time parameter, in samples: With the FlexDC, each sample is 122[μs]. WT defines the wait time between consecutive motions. Upon BG, the controller generates a motion toward AP, waits WT samples and then generates motion toward the original position, where it waits again WT samples, and so on.
- KR: Kill Repetitive command. Unlike a standard PTP motion, a Repetitive PTP motion is not finished unless stopped by the user or any fault or limitation. While AB and ST acts just as for a standard PTP motion, KR stops the repetitive sequence, completing the current PTP motion and only then stopping. A Repetitive PTP motion is started just as a standard PTP motion but with SM=1, instead of SM=0. This means that the basic motion mode is still a PTP motion (MM=0) but it has a special modification, identified by SM=1.

Notes:

- *Each motion segment within a repetitive motion is treated as a standard PTP motion. The only difference is reflected in the SR parameter, bit 4 (In Repetitive PTP motion). In addition, when a motion segment is finished and the motion is "paused" for WT samples, a dedicated bit in MS identifies this status (bit 6).*
- *Modifying AP on-the-fly modifies the target position of the current segment but does not affect the second target position (the "back" motion).*
- *In the FlexDC the repetitive motion is also supported under STEP mode (MM=8).*

3.3 Jogging – JOG (MM=1, SM=0)

3.3.1 Description

In this mode the controller calculates a standard acceleration profile, using the user specified acceleration (AC) toward the user specified speed (SP).

This speed is kept constant until the motion is stopped by a user command.

In case of an ST (Stop) command, the controller calculates a deceleration profile, using the user specified deceleration (DC).

The motion's direction is set according to the sign of the SP (Speed) parameter.

3.3.2 Starting a Jog Motion

Communication Clauses	Description
MO=1	Enabling the servo loop, motor ON
MM=1;SM=0	Setting Jogging motion mode
AC=500000	Assigning a value for the acceleration, [counts/sec^2]
DC=200000	Assigning a value for the deceleration, [counts/sec^2] Used when stopped or when changing SP on the fly.
DL=1000000	Assigning a value for the Limit DC, [counts/sec^2]
WW=0	Defines no smoothing.
SP=50000	Assigning a value for the speed, [counts/sec]
BG	Begin the motion

3.3.3 Monitoring a Motion

See section 3.1.2 above, Part II.

3.3.4 Stopping a Motion

A jogging motion is, theoretically, an infinite motion. It stops only as a result of a user command or due to some fault, limitation or protection.

A jogging motion can be stopped by one of the following communication clauses:

Communication Clauses	Description
AB	Aborts the motion immediately (DP remains as its last value).
ST	Stops the motion with deceleration (using DC) to zero speed.
MO=0	Disables the motor, effectively stopping any motion.

Any software or hardware fault, limitation, or protection also immediately aborts or stops the motion (depending on the fault or limitation type).

3.3.5 On The Fly Parameters Change

The following parameters can be modified on-the-fly during an active jogging motion:

Communication Clauses	Description
SP	Starts an acceleration or deceleration toward the new SP value. The New SP value can have a different sign from the previous SP value.
AC, DC	Affects any following motion toward a new SP value.

3.4 Gearing Motion Modes

3.4.1 Position Based Gearing (MM=2)

3.4.1.1 Description

Gearing (or electronic gearing) motion is refer to a motion mode where an axis follows another axis position with a pre-defined (fixed) ratio. The FlexDC supports position gearing motion mode for X and Y axes only.

The position gearing is implemented based on a master DP follow method. In this method, the follower axis is slaved to a (user selected) Master Axis Desired Position (i.e. master's DP, not its actual encoder position PS). This method allows for very accurate multiple axes vector motions, with one axis being used as a master, while the other axis can be slaved to it's reference position (i.e. to the master's theoretical profiler output).

Note:

- *The master axis can be set to motor ON or OFF (i.e. MO=0). In the latter case, the master's DP=PS, so using a disabled axis as a master axis provides true encoder position tracking.*

The following dedicated parameters are used for Position Based Gearing Motion:

- "ME" – Master Encoder or Axis. This parameter defines which axis is the master axis for a given slave gear motion. The "ME" parameter can select between the following encoder inputs: ME=0 for X Axis Encoder, ME=1 for Y Axis Encoder.
- "FR": Following Ratio. This parameter defines the slave's following ratio in relation to the master's axis ("ME") reference position ("DP"). "FR" can be any number in the range of: [-2,147,000,000 ÷ 2,147,000,000]. As noted above, "FR" is an integer number scaled to 8.24 format. i.e., "FR=16,777,216" means a following ratio =1.0.

Note:

- *The "FR" parameter is using a 32 bit, 8.24 format scaling resolution, to allow ratios of up to: $\times \pm 128$, and : $\times \pm {}^{1}/_{16,777,216}$.*

The slave axis reference position is relative to the master's and slave's initial position when the slave axis was initially commanded to actually begin the Gearing Motion.

Gearing motion is initialized like any other motion. This means that first the motion parameters and mode should be set, and then a valid "BG" (Begin Command) should be given. Upon issuing a "BG" command to an axis in MM=2, first the master and slave initial positions are locked, and then the axis enters a motion state where its reference is calculated according to the following equation:

$$\boldsymbol{SlaveDP} = \frac{\left(\boldsymbol{MasterDP} - \boldsymbol{MasterInitDP}\right) \cdot \boldsymbol{FR}}{\mathbf{16,777,216}} + \boldsymbol{SlaveInitDP}$$

Notes:

- *For an axis in gearing motion mode (the slave), all other motion profiler parameters (i.e. "SP", "AC", etc.) are ignored.*
- *Users should avoid altering a master axis "DP" (by a issuing a "PS=" command to the master axis) while it is connected to a slave axis that is in motion, to avoid position Step Commands to the slave and possible a high error faults.*
- *Although "FR" can be change during motion, doing so results in a slave step command, which may cause a high error fault.*
- *When an axis is commanded to begin a motion in MM=2, it immediately enters the motion with the reference as defined above. No acceleration profile is generated for cases where the master axis is already in motion.*
- *Like jogging, gearing motion is also theoretically an infinite motion. It stops only as a result of a user command or due to some fault, limitations or protections.*
- *If a gearing motion is stopped (by a user "ST" command), or by other faults such as hardware or software limits, the slave axis starts to decelerate using the relevant deceleration parameters: "DC" for normal stop commands ("ST") and "DL" for limit stop conditions. In this case of course, the axis is "losing" the master's tracking.*
- *In Gear Motions "WW" (the smoothing parameter) must be "0", since the slave is directly following the master DP according to the equation described above. A "WW" value other than "0" does not affect normal tracking, but causes a position step command when a stop command is given.*
- *Like in all other motions, an "AB" (abort motion) command results in immediate stop of motion without any deceleration profile.*
- *Due to an implementation limitation, only when X is following Y, one sample time delay (122 micro-sec in FlexDC) is present in the generated slave axis (X) reference profile, related to the master profile (Y).*

3.4.1.2 Starting a Position Based Gearing Motion

Communication Clauses	Description
YMO=1	Enabling Y Axis servo loop, motor ON
YMM=2;YSM=0	Set Y axis to Position Based Gear Mode
YME=0	Set Y Master Axis As X (Y follows X)
YFR=1,048,576	Set Following Ratio to $^{1}/_{16}$.
YBG	Start Y Motion (following the X axis)

In this example, the Y axis is commanded to follow the X axis's reference position, with a ratio of $^{1}/_{16}$. Note that typically, when an axis is intended to operate in gear mode, the following axis is first being enabled and enters motion (BG), and only afterwards the master axis is commanded to move. Starting a gearing motion (BG with MM=2), where the master axis is already in motion results in a velocity command step to the following axis.

3.4.1.3 Monitoring a Position Based Gearing Motion

See section 3.1.2 above, Part II.

3.4.1.4 Stopping a Position Based Gearing Motion

As noted above, gear motion is, theoretically, an infinite motion. It stops only as a result of a user command or due to some fault, limitation or protection.

A gear motion can be stopped by the following communication clauses:

Communication Clauses	Description
AB	Aborts the motion immediately (DP remains as its last value).
ST	Stops the motion with deceleration (using DC) to zero speed. Note that immediately after issuing the “ST” command, the slave axis stops following the master, and starts an autonomous stop profile motion towards zero speed.
MO=0	Disables the motor, effectively stopping any motion.

Any software or hardware fault, limitation, or protection also immediately aborts or stops the motion (depending on the fault or limitation type).

Note:

- *In gear motion, any fault condition acting on the master axis, does not directly affects the following (slave) axis. This means that the following axis remains linked to the master DP, regardless of the master's motion status or motor status. For example, if a master axis is disabled due to a high error condition, its motor does not turn OFF, but the following axis still is in motion condition, and keeps following the disabled axis encoder, even after it is stopped.*

3.4.1.5 On-The-Fly Parameters Change

An axis during gear motion is not affected by any of the normal profiler motion parameters (e.g. SP, AC, etc.).

Although the following ration ("FR") can be modified during motion, it is not recommended to do so, as it results in a position and possibly also velocity reference steps.

3.5 Joystick Motion Modes

3.5.1 Velocity Based Joystick Motion Mode

3.5.1.1 Description

This mode is very similar to the jogging mode. However, instead of jogging in the user specified SP value, the jogging speed is taken from the analog input (assuming it is connected to a Joystick or any other source of analog voltage).

The analog input parameter AI is used instead of SP. All other parameters (AC, DC etc.) are used exactly as for jogging mode.

3.5.2 Position Based Joystick Motion Mode

3.5.2.1 Description

This mode is very similar to the standard PTP mode. However, instead of using the user specified Absolute Position (AP) parameter as the target position, this mode uses the Analog Input (AI) parameter as its target position.

Since a standard PTP mode supports on-the-fly modification of the AP parameter, this mode automatically supports the changes of the AI during the motion, practically tracking them with the user specified acceleration (AC) and speed (SP) parameters,

These parameters needs to be high enough to enable good tracking on the joystick motions (variations of the AI parameter) but low enough to avoid “high-frequencies” motions.

It is important to note that when this mode is activated using the required MM and SM values, the AP parameter is continuously and internally assigned with the AI value.

3.6 Position Step Motion (MM=8, SM=0 or SM=1)

3.6.1 Description

In this mode the Desired Position (DP) is assigned with the Absolute Position (AP) immediately after the Begin (BG) command. The profiler does not generate any motion profile and the AC, DC and SP values are ignored. The theoretical motion time in this mode is “0” by definition (True Step command).

This mode is useful for the measurement of the closed loop step response and bandwidth. It is generally not used in practical applications since it generates infinite acceleration and jerk. MM=8 can be combined with SM=1 to generate repetitive step motions.

Note that you can also use the Relative Position (RP) parameter. Assigning a value to RP modifies the value of AP properly.

The value of the step should be smaller than ER to avoid High Error fault. In addition, high step values can cause oscillations due to the non-linearity’s (especially saturation) which are an inherent part of the control loop.

3.6.2 Starting a Step Motion

Communication Clauses	Description
MO=1	Enabling the servo loop, motor ON
MM=8;SM=1	Setting Position Step motion mode (Repetitive Mode)
AP=100	Assigning an absolute target position, [counts]
RP=30	**or**, assigning a relative value for the target position
BG	Begin the motion

Note that the FlexDC also supports repetitive Step Motions. Similarly, this can be done by setting SM=1 instead of SM=0 in the above sequence. WT is used as the delay time between each two consecutive motions.

3.6.3 Monitoring and Stopping a Step Motion

Note that the step motion mode is very short (one sample time). As a result, it is practically impossible to monitor the state of this motion.

In addition, a step motion does not affect the EM parameter, which remains at the same value as it was before the BG command.

Since the step motion is very short, it is not practical to stop it after a BG command.

If a repetitive step motion is commanded, the user should use the KR (Kill Repetitive) command, much like a normal PTP Rep motion.

3.7 Profile Smoothing in the FlexDC

The FlexDC supports an advanced, symmetric S-curve like profile smoothing algorithm. The smoothing is controlled by the "WW" parameter.

The "WW" can be set to 0 to avoid any profile smoothing. In that case the generated position velocity profile is pure trapezoidal (or triangular).

If the "WW" is set to 12, the smoothing is set to its maximal value. In that case the generated profile has full smoothing, and the velocity trajectory is not a pure trapezoidal.

The "WW" parameter is used by the controller as a power of 2 coefficient for the smoothing time value. For example, WW=6 means that smoothing is done over a period of time of 2^6 sample time = 1000*2^6/2^13 ~ 8msec.

Setting WW=12 to its maximal smoothing value of 2^12, results in a 0.5 sec.

The following figures show two simple profiles generated in similar motion parameters, with different smoothing values.

For both motions, the following general parameters are used:

AC=DC=1,000,000

SP=100,000

AP=100,000

In one case no smoothing is used (WW=0), and in the other full smoothing is defined (WW=12).

Figure 7 below shows the motion profile with full smoothing implemented in the profile. Note the smooth velocity profile (the upper window in red).

There are no “sharp” corners in the generated velocity profile (like in Figure 8 for instance).

The resulted acceleration profile (not shown in the graph) is continuous and does not have any sudden “step” changes.

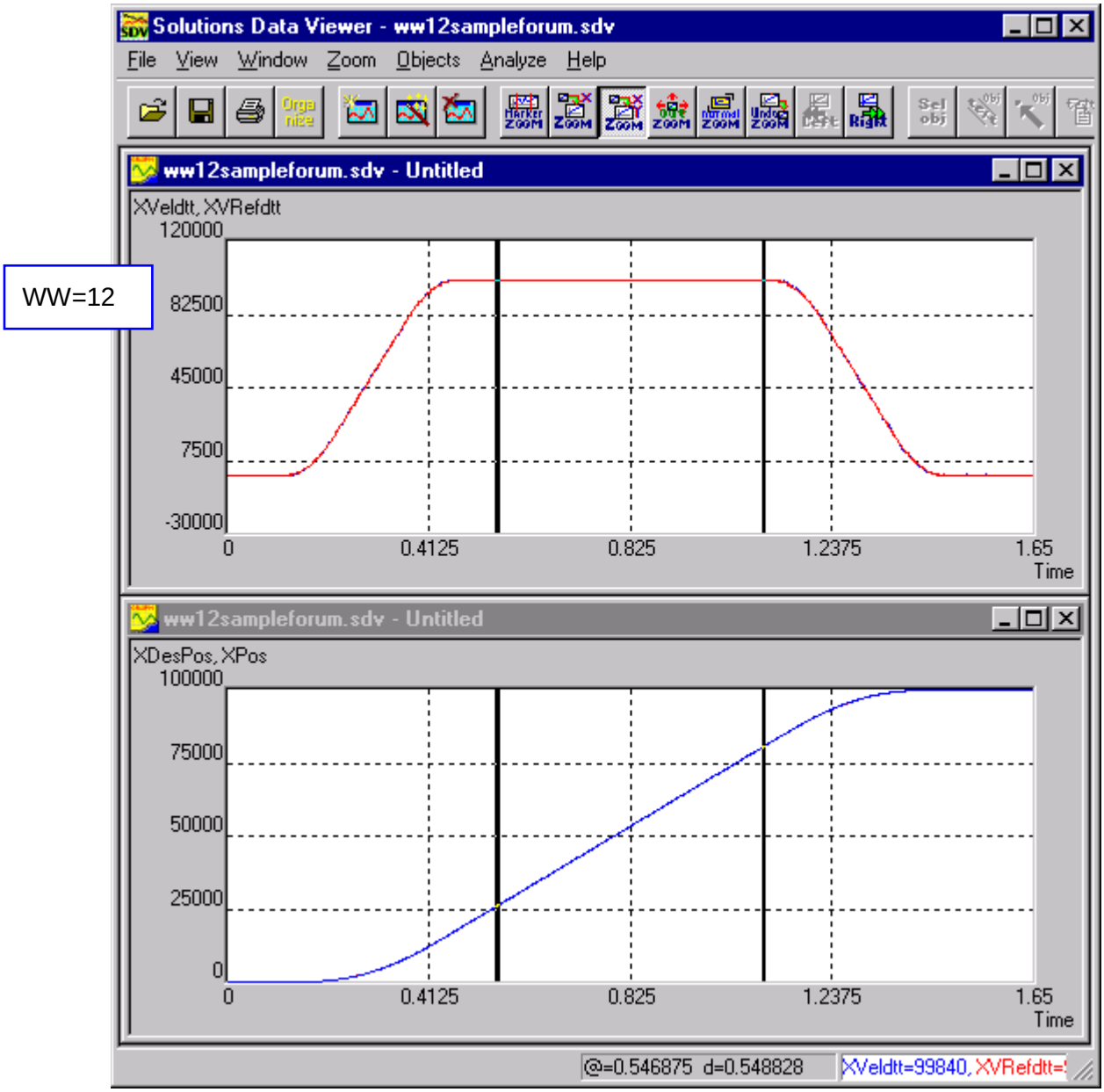


Figure 7: Typical motion profile with full smoothing

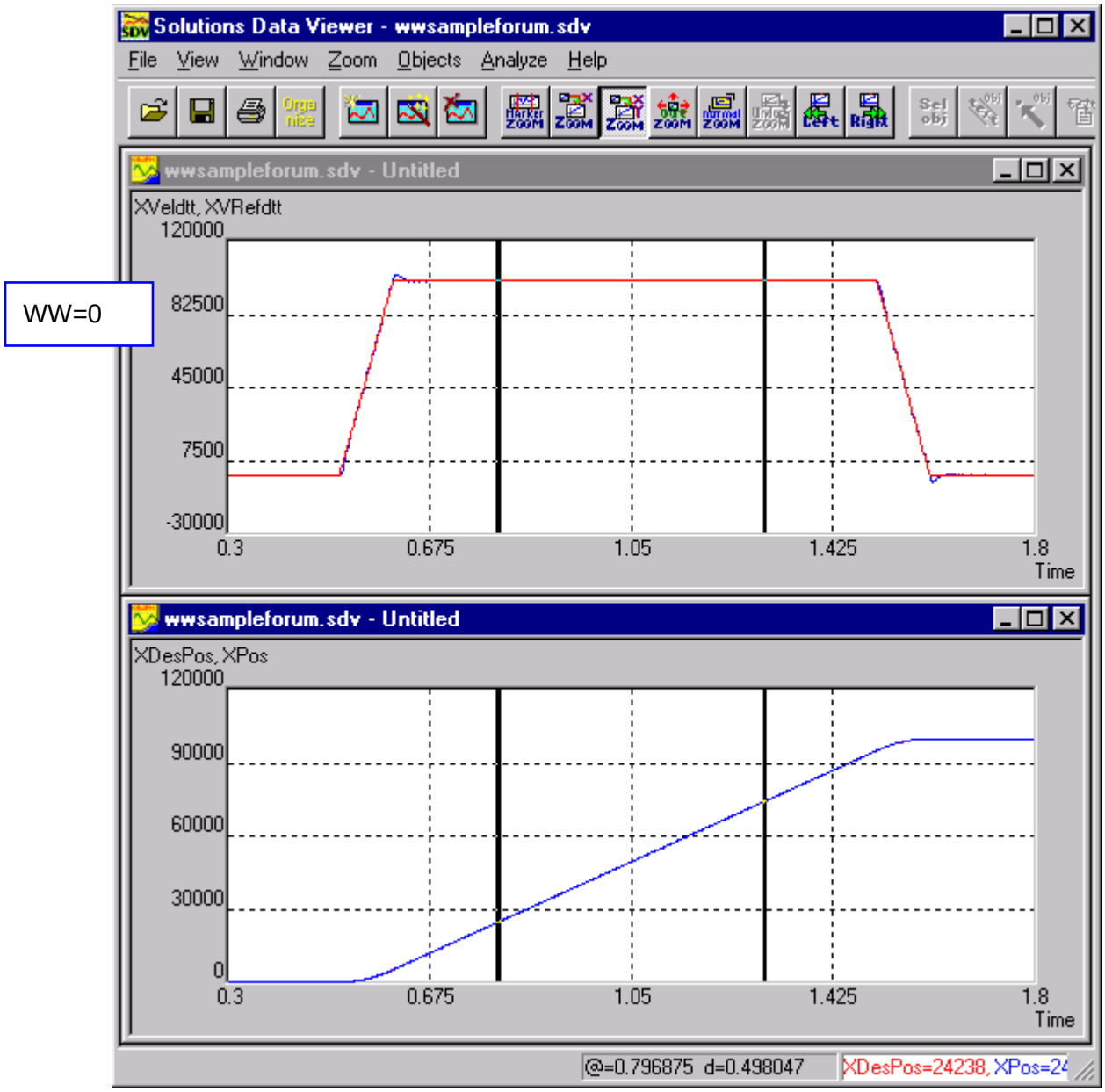


Figure 8: Typical Motion Profile with no Smoothing

Note that profile-smoothing implementation does not imply any numerical limitations, and does not include any “minimal motion time” limit, which may be implied by the use of the smoothing itself.

The user should be aware that theoretically, a smoothed profile takes longer time to complete than a similar trapezoidal profile with no smoothing.

The actual time difference between the non-smoothed theoretical trapezoidal profile to the smoothed one depends on all motion profile parameters (SP, AC, DC and the motion distance of course).

In any case, the maximal time difference does not exceed the overall smoothed period (2^WW sample times).

4 The Control Filter

4.1 General

The FlexDC Filter is based on Position over Velocity Loop PIV Filter (see Figure 10).

In the following sections the Linear Filters equations and Non-linear Algorithms are described in details.

4.2 Linear PIV Filter Equations

4.2.1 PIV Filter Mode

In closed loop operation in PIV mode, the control loop structure can be considered as divided into two separated loops: an external position loop cascaded over an internal velocity loop.

The velocity loop linear PI filter in PIV mode is shown in Figure 9.

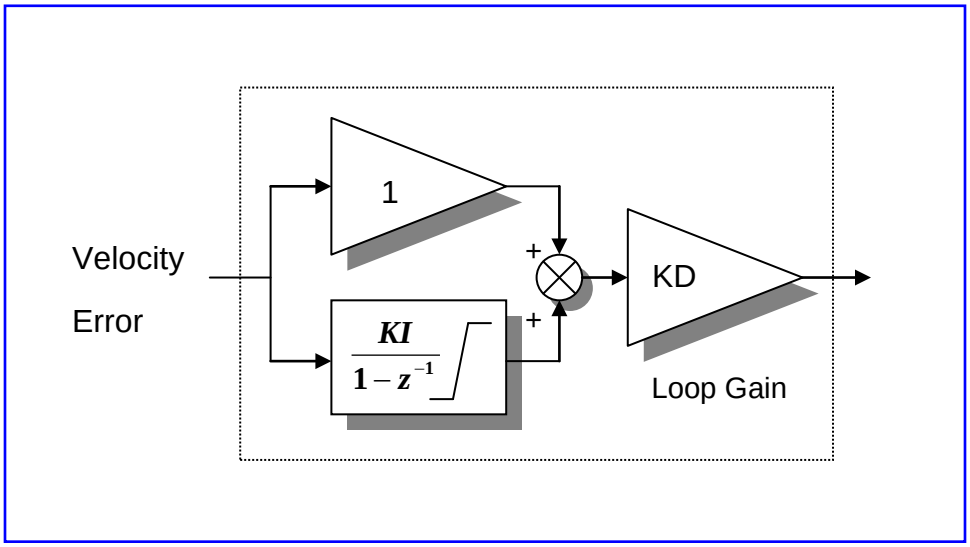


Figure 9: Velocity PI Controller

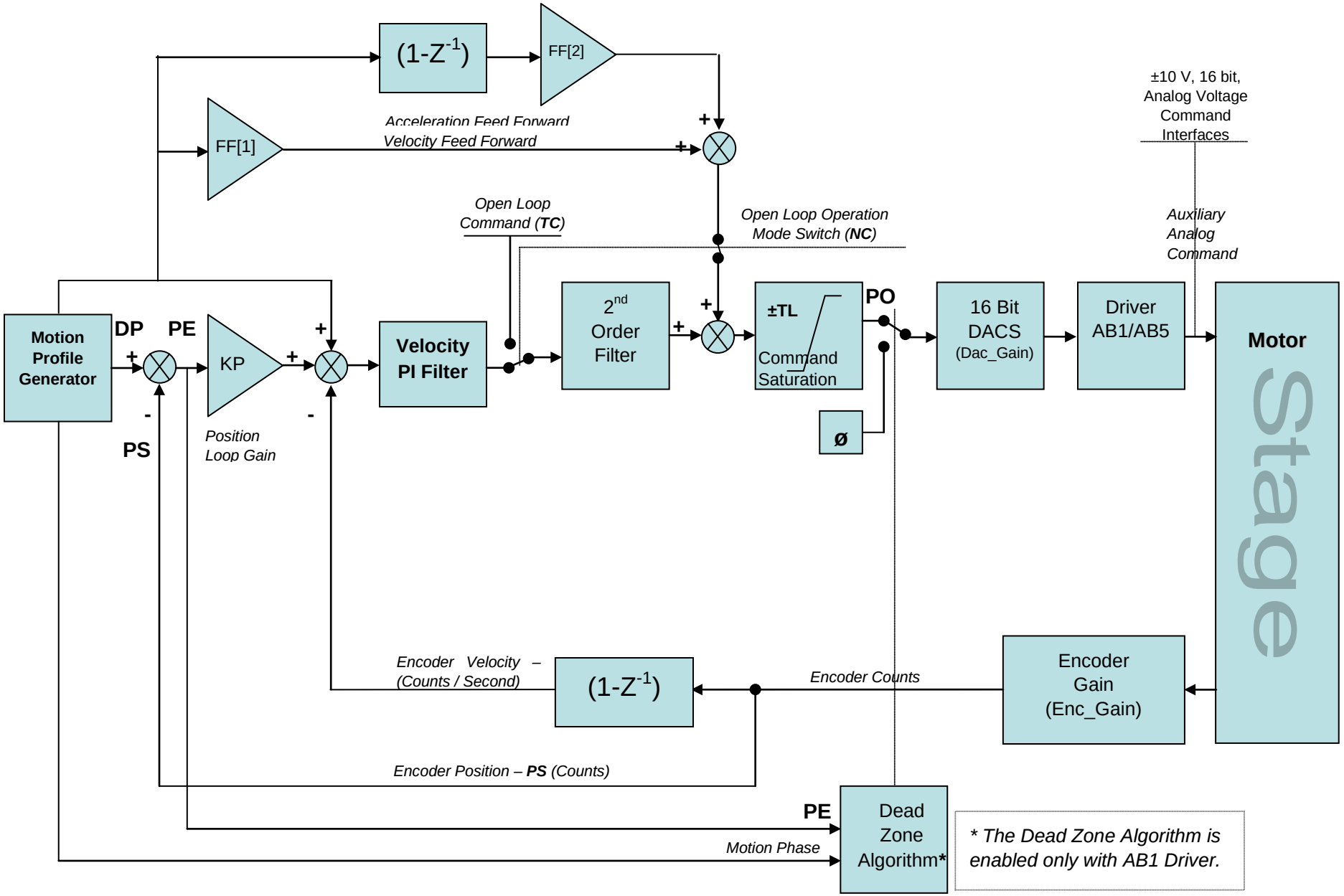


Figure 10: The FlexDC PIV Filter

The linear filter equations in the PIV Filter Mode are:

$$PE_k = DP_K - PS_K$$
$$VC_k = PE_k \times KP + (DP_K - DP_{K-1}) \times 65536$$
$$VE_k = VC_k - (PS_K - PS_{K-1}) \times 65536$$
$$VE'_k = VE_k \times KD$$
$$U_k = \frac{VE'_k}{65536} + KI \times \sum_{i=0}^{k} VE'_k$$
$$PO_k = Sat(TL, U_k \times SecondOrderFilter)$$

where:

- ***DP, PS*** *and* ***PE*** *are the desired position, actual position and Position Error (similar to PID mode).*
- ***KP*** *is the position loop gain.*
- ***KI*** *is the velocity loop Integral term gain.*
- ***KD*** *is the velocity loop overall gain multiplier.*
- ***VC*** *is the velocity loop reference command. Note that* ***VC*** *includes an inherent (not controlled by the user) velocity command Feed-Forward element, represented in the second equation above by:* DP_K-DP_{K-1}.
- ***VE*** *is the internal velocity loop error. The velocity loop feedback is currently used as a simple numeric derivative of the position reading, represented in the third equation above by:* PS_K-PS_{K-1}. *Both* ***VC*** *and* ***VE*** *are internal software variables and can not be accessed from the communication.*
- ***U*** *is the Velocity PI filter output.*
- ***TL*** *is the output command saturation value.*
- *The 2nd order filter block is filter high order low-pass filter.*
- ***PO*** *is the final control loop output. This value is converted to the analog output command for the external driver using the 16 bit DAC in the system.*

The filter equations in this case can also be written in a Z transform transfer function equation as follows:

$$U = PE \times \left(KP + (1 - z^{-1}) \times 65536\right) \times KD \times \left(\frac{1}{65536} + \frac{KI}{(1 - z^{-1})}\right)$$

Note that the final closed loop transfer function (2 zeros and an integral), has different parameter scaling, and an isolated parameters form. This can be considered as a more convenient filter form, as one can note that the filter has 2 zeros, separately effected by KP and KI, an integral, and total loop gain KD (actually the velocity loop gain).

Another benefit in that form is that one can operate the closed loop system with KP=0 (no position feedback) to tune the velocity loop performances only, and then use the KP gain to control the position loop gain (and resulted bandwidth).

4.2.2 Position Error Calculation

The Position Error variable PE is a read only parameter, updated by the real time control loop, and computed by:

$$PE_k = DP_K - PS_K$$

where, as noted above: ***DP, PS*** and ***PE*** are the desired position, actual position and Position Error, all in encoder count units.

The Position Error is always “0” by definition whenever the servo is OFF (MO=0), since the servo controller automatically updates the current desired position “DP” to be equal to the actual position “PS”.

During all Servo ON modes (MO=1), in both open and closed loop cases, the real time control loop checks the current Position Error value (“PE”) and compares it to the maximum allowed Position Error (ER). Whenever PE > ER the real-time loop automatically disables the motor and indicates the error reason as High Error fault.

In the FlexDC, the maximum ER value can be as high as 8,000,000 counts.

4.3 High (2nd) Order Filters

The FlexDC includes a digital 2nd order filter. The filter can be enabled or disabled using a special dedicated new parameter: CA[13].

- When CA[13]=0 the 2nd order filter is disabled in all modes.
- When CA[13]=1 the 2nd order filter is enabled in all modes.

The 2nd order filter equations are:

$$Y_k = \frac{U_K \times a_0}{1 - b_1 \times Y_{K-1} - b_2 \times Y_{K-2}}$$

or

$$Y_k = a_0 \times U_K + b_1 \times Y_{K-1} + b_2 \times Y_{K-2}$$

where:

U and ***Y*** are the filter input and output signals, and

a0, b1, b2 are the filter constants.

The filter parameters are user defined, and are set in by a special set of dedicated new parameters: CA[7], CA[8], and CA[9] with the following scaling:

CA[7] = *a0* x 65536 x 16384.

CA[8] = b1 x 65536.

CA[9] = b2 x 65536.

With the Nanomotion Shell Application, the user can easily and automatically set filter variables. The Shell provides a utility that converts standard Frequency and Damping values to the controller filter form parameters scaling. The Shell is using a standard Z transform for the conversion.

Note:

- *The 2nd order filter is present in both closed and open loops. The user can test the operation of the filter in open loop, and record the step response of the filter. This can be done (when the 2nd order filter is enabled) by switching to open loop mode (NC=1), issue a torque command (TC=XX), and record the driver command signal.*

4.4 Output Command (DAC Out)

The FlexDC has a 16 bit DAC Output.

- +10V is represented by +32,767 and
- –10V is represented by –32,767.

4.5 Encoder Gain

The controller counts quadrature encoder pulses. This implies a feedback gain. For example, a typical linear system with an encoder of 0.1µm resolution, mounted on the linear stage, the encoder's gain is as follows:

Enc_Gain = 10^7 counts/m

4.6 Non-Linear Elements

The actual control filter structure includes the following non-linearities:

- The filter command output is saturated to the value of the "TL" (Torque Limit) parameter. The output command saturation is active at all times in all modes. The software range limit for "TL" is 0 ÷ 32,767 in DAC [LSB] units or from 0 to 10 V to the Driver command. It is recommended to set TL to 32767 full drive command.
- When working in closed loop operation only, the filter Integral term output is also saturated to the value the IS (Integral Saturation) parameter. The software range limit for "IS" is 1 ÷ 32767 in DAC [LSB] units. It is recommended to set the Integrator limit IS to 16500 (approximately half of the full drive command).
- To the value of "PO" (the final filter signal output, after the 2^{nd} order filter calculations) an offset value defined by "DO" (DAC Offset) is added in order to compensate analog output voltage offset. Although the software range limit for "DO" is ±32,767 in DAC [LSB] units, it is usually not required to use values more then few hundreds. Note that by using high values of "DO", a non-symmetrical analog output range can be resulted. The final DAC command is always protected from roll over beyond 16 bit value. It is recommended to set DO to "0".

- The encoder has a finite digital resolution, which also implies a non-linear quantization effect.
- Non-Linear Gain Scheduling – see the next section for more information.

4.7 Filter Gain Scheduling

The FlexDC software has a built in control filter gain-scheduling logic. The gain-scheduling logic may be used in order to improve the settling performances of a system (mainly to reduce settling times).

This is done, simply, by changing the filter's constants (KP, KI, KD) for a short period of time after a motion is completed. The user can define the period (after previous end of motion condition) in which the gain-scheduling is effective.

The following parameters can be used by the user in order to operate the gain-scheduling feature:

- KP[2] is the parameter replacing KP (= KP[1]) when gain-scheduling is active.
- KI[2] is the parameter replacing KI (= KI[1]) when gain-scheduling is active.
- KD[2] is the parameter replacing KD (= KD[1]) when gain-scheduling is active.
- CA[4] is the gain-scheduling period, in servo sample time.

The gain-scheduling is active (i.e. KP[2], KI[2], KD[2] are used) after a motion is fully completed (Motion Status bits are not in motion), for a period of CA[4] sample times. If before that a new motion has begun, the gain-scheduling is immediately disabled.

To disable the gain-scheduling, the user can simply set KP[2]=KP, KI[2]=KI, KD[2]=KD, and/or set the period CA[4]=0. Both disable the feature.

The user should avoid using too high parameter settings to avoid the system losing stability when the gain scheduling is active. It is also not recommended to use this feature when very high position errors are reached during final motion acceleration phase.

4.8 AB1A Driver Special Algorithms

4.8.1 Dead Zone Algorithm

This algorithm improves the settling time, taking full advantage of the intrinsic motor friction. The algorithm sends zero command to the driver when the position approaches the target within the Dead Zone Min CA[36] in encoder counts unit. When the Position Error increases above the Dead Zone Max CA[37], the controller starts “servoing” again. Typically, in a system with encoder resolution of 0.1µm Dead Zone Min is between 1 to 2 counts and Dead Zone Max is between 4 to 10 counts.

Note:

- *This mode is active only when not in motion. This mode can be enabled or disabled by bit 1 in CA[33] (0 based).*

4.8.2 Feed-Forward Algorithm

This algorithm "zeros" the Feed-Forward Velocity value if the absolute distance between the current position (PS) and the final destination (AP) is smaller or equal to the value of CA[38]. The value depends on the total moving mass and encoder resolution. Typically, this value equals to 20-50µm. This algorithm is active in single Point-To-Point motions mode only (i.e. MM=0, SM=0).

Note:

- *This mode can be enabled or disabled by bit 0 in CA[33] (0 based).*

4.8.3 Offset Algorithm

The offset algorithm improves the start motion time by reducing the motor dead band.

If desired velocity is positive - CA[34] value is added to driver command.

If desired velocity is negative - CA[35] value is added to driver command.

Note:

- *The offset is added before the 2nd order filter and FF.*
- *This mode works in repetitive motion.*
- *This mode can be enabled and disabled by bit 2 in CA[33] (0 based).*
- *It is user's responsibility to verify that the range of the offset is correct (0-3200 max).*

4.8.4 UHR Algorithm

The UHR algorithm takes the Velocity Filter output and creates a PWM command.

The cycle of the UHR, can be defined in CA[39]:

(CA[39]=8) defines 100% PWM.

Notes:

- *If the CA[39] value is bigger than 8, command to driver is normal PIV output.*
- *If the CA[39] value is smaller than '0', command to driver is always '0'.*
- *The UHR algorithm does not affect driver offset for analog output.*
- *The UHR algorithm works in both open and closed loops. (CA[39]=0) defines 0 PWM.*

4.9 AB5 Driver Brake Mode

The FlexDC Motion Controller has the Brake Mode when working with AB5 driver. In Brake ON Mode the driver disconnects the power supply to the motor; however servo is still active. As the motor is turned OFF, it consumes no power and the EOP can be extended. Refer to the "AB5/AB51 Driver User Manual" for detailed information.

Note:

- *The combination of Brake ON while the driver is Enabled poses a conflict and the user must consider the transient upon returning to the Brake OFF Mode.*

4.10 Acceleration and Velocity Feed-Forward

The FlexDC supports reference command Feed-Forward features.

- Command Acceleration Feed-Forward (Acc-FF) is supported in closed loop modes. The Acceleration Feed-Forward gain is controlled by the FF[2] parameter. FF[2]=0 means no acceleration Feed-Forward is used. The Acceleration Feed-Forward Gain (FF[2]) is working on the profile acceleration in counts/sec2 / 219 units. It is recommended to set FF[2]=0.
- Command Velocity Feed-Forward (Vel-FF) is currently supported in closed loop control mode only. The Velocity Feed-Forward gain is controlled by the FF parameter (FF[1]). In most of the cases FF[1] should be set to "0".

In both cases, the resulted Feed-Forward value is added to the filter command output, in DAC [LSB] units.

4.11 Open Loop Operation

The FlexDC supports a dedicated open loop operation mode. In this mode the user can directly set the value of PO, without the closed loop control filter, and regardless of the system position readings or the position or velocity errors.

Note that although under open-loop mode, the High Position Error protection mechanism of the controller is still active (see chapter 5, Part II). TL always saturates the command, even when operating in open loop mode. If the maximum "ER"=8000000 is not enough to operate the open loop, it is recommended to disconnect the encoder plug from the FlexDC.

The method to activate this mode is to use the NC parameter to disable the closed loop operation (set NC=1, in motor OFF, and then set motor ON) and to use the TC (Torque Command) parameter to set the desired PO value.

Because the offset DO is always added to the PO, the actual PO value equals to:

PO = TC + DO.

As the 2nd order filter is applied also under the open loop mode, it is possible to record the step response of the filter. Use open loop operation and record the Driver Command signal (see section 4.3, Part II).

4.12 Summary of all Control Filter Related Parameters

Table 6 summarizes all servo loop related parameters of the FlexDC supported.

Keyword	Description
MO	Motor ON – Enables (MO=1) / Disables (MO=0) the servo loop.
NC	No Control – Enables (NC=1) / Disables (NC=0) open loop mode.
TC	Torque Command in open loop mode.
TL	Torque Limit – Limits the D2A command – All modes.
IS	Integral Term Saturation of PIV control filter.
PO	The final control filter output command value.
DO	The control filter offset calibration parameter.
CG[Bit3]	Configuration Bit controlling set to “0” (PIV mode).
KP,KP[1]	Proportional position gain.
KI,KI[1]	Integral gain, in velocity loop.
KD,KD[1]	Derivative gain, in velocity loop.
KP[2]	KP Gain when gain-scheduling is active.
KI[2]	KI Gain when gain-scheduling is active.
KD[2]	KD Gain when gain-scheduling is active.
CA[4]	Gain-scheduling period.
CA[7]	2^{nd} order filter A0 gain.
CA[8]	2^{nd} order filter B1 gain.
CA[9]	2^{nd} order filter B2 gain.
CA[13]	2^{nd} order filter Enable (if “1”) Disable (if “0”) flag.
FF,FF[1]	Velocity Feed-Forward Gain.
FF[2]	Acceleration Feed-Forward Gain.

Table 6: Control Filter Parameters

5 Faults Protections and Limits

The FlexDC includes various protection mechanisms and status report parameters, which ensure safe operation and easy troubleshooting.

The protective mechanisms are divided into two groups: protections and limitations.

- **Protection** refers to the detection of a fault condition and the response to this condition (generally disabling the servo).
- **Limitation** refers to an algorithm which continuously monitors and limits (saturates) a value, keeping it from reaching a fault condition.
- **Fault** represents a list of conditions which are detected and responded to with a proper protection function.

Some of the protections are implemented directly by the hardware, ensuring safe, fast and immediate response, while some are implemented by software, providing user control of the protection behavior.

All detected faults result in immediate “Servo OFF” condition. Analog signal commands are reset to “0” voltage, and the drivers are immediately disabled.

The FlexDC controller detects the following faults:

- Driver Fault (via the Fault Input) – refer to the "Fault Output" and "TP (Termal Protection)" sections in the "AB5/AB51 Driver" User Manual.
- High Position Error.
- Encoder Signal Error – two types of encoder error detection.
- Motor Stuck Condition.

The FlexDC includes the following protections:

- Verification of correct firmware and FPGA versions after power on.
- Forward Limit Switch – stops any on going motion in the relevant direction.
- Reverse Limit Switch – stops any on going motion in the relevant direction.
- High Position Software Limit – stops any on going motion in the relevant direction.
- Low Position Software Limit – stops any on going motion in the relevant direction.

The FlexDC includes the following limitations:

- The peak driver command is limited, usually to limit the max Force/Velocity command to the motor. Driver command limitation has two different parameters, TL (which is the ultimate command saturation limit), and IS which can (separately from TL) limit the Integral value. This is needed in some cases to improve dynamic responses. It should be noted that the value of TL overrules the value of IS (see chapter 4, Part II, for details).

The following sections provide a more detailed description of the faults, protections and the controller response in each case.

5.1 Driver Faults and Abort Input

Driver Fault is a condition indicating that something is wrong with the motor power/driver connected to the controller. The Driver Fault is an actual hardware signal line that the driver outputs. This signal is continuously monitored by the controller real time servo loop, at the main control sample rate of 8 kHz. If the real time software detects that this line is active, the servo loop axis related to the relevant faulted driver is immediately disabled.

There is a separate, independent Driver Fault Input line for each one of the controller axes. When an axis is disabled by a Driver Fault, the controller automatically switches to the Servo OFF (MO=0) condition in that axis. In this condition the controller's driver inhibit output is activated, and the analog command is immediately switched to "0" value.

The user can switch the actual logic of the Driver Fault line separately for each axis. This enables to support any type of Driver Fault electrical and logic interface (active high or active low). See the CG (axis configuration word) command for more information in chapter 7, Part II.

The following parameters reflect the Driver Faults and Abort conditions:

Controller State	Description
MO is set to "0"	The motor ON parameter is reset to "0".
EM is set to "6"	Last Motion End Reason is "6"- Motor Fault.
MF is set to "1" for DRV **MF is set to "2" for ABORT**	Motor Fault Reason is Driver Fault Input. Motor Fault Reason is Abort Input.
IP[24] is "1" for XDrv Flt **IP[25] is "1" for YDrv Flt**	The relevant bit in IP (the Input Port Word) is set active (high). Bit 24 for X Driver Fault, Bit 25 for Y.
IP[28] is "1" for ABORT	Bit #28 is set high is the Abort input is Active (no current through the Abort lines).

5.2 Software Generated Faults

The FlexDC real time servo loop software can generate the following faults:

- High position loop error.
- Encoder Signal Error.
- Motor Stuck Condition.

Each one of the above axis related fault conditions generates similar result to a Driver Fault condition. The specific axis is immediately disabled, and the relevant software status bits are updated.

5.2.1 High Position Error

This error occurs when the servo loop position error is too high.

The Position Error "PE" is continuously compared to the maximal allowed error value "ER". Whenever "PE > ER" the axis is disabled.

The High Position Error protection is active at all times when a servo axis is enabled (i.e. when MO=1). This means that the Position Error is also monitored when the axis is in open loop modes. The max allowed positioning error is 8,000,000 encoder counts.

High Position Error fault is reported by "MF=3".

5.2.2 Encoder Signal Error Protections

The FleXDC hardware supports two types of encoder signals error conditions:

- **Encoder A Quad B Error:** This error is detected when the controller encoder hardware interface detects that both “A” and “B” encoder lines are changed simultaneously. In normal A quad B encoder operation this is an invalid condition. The encoder signal lines are sampled by the hardware at a very high rate, and If in a single sample event both “A” and “B” changes their state, the error is asserted.
- **Encoder Disconnected Line Error:** This error is detected when the controller encoder hardware interface detects that one of the following: “A”, “!A”, “B”, “!B” signals are not connected. The condition is detected by sampling all signals, and evaluating the following state: “(A== !A) | (B == !B)”. If the state is true for more then 4 consecutive servo samples, the error is stated.

The second error condition (disconnected line), requires a full differential encoder interface to be used. The protection cannot be used in single ended line encoders. Note that only the “A+/A-” and “B+/B-” lines are sampled for errors. There is no implementation for Index disconnected line detection. The user can enable or disable the encoder error detection by a dedicated bit in the axis configuration word “CG". Encoder Error faults (when enabled), are reported by special code in the “MF” keyword (the Motor Fault Cause).

5.2.3 Motor Stuck Protection

The purpose of the Motor Stuck Protection is to protect the motor from continuous high command operation.

When the command reaches the TL value and the Velocity of the motor is "0", after 1 second, the controller automatically disables the axis.

This protection operated only in closed loop.

Motor Stuck Fault is reported by “MF=4”.

5.3 Software Protections – (Non Fault Conditions)

The following software protections are managed by the controller without generating a fault condition. This means that the servo axis stays enabled, even though the protection may be active.

- **FPGA Version:** During the controller boot process, the firmware reads the FPGA version, and verifies that the current version matches the firmware version. An error is indicated if the version does not match. The error is indicated by the CPU LED, blinking eight times, during the boot process. If the FPGA version error occurs, please contact Nanomotion experts for further instructions.
- **CAN Hardware Initialization Failure:** During the controller boot process, the firmware initializes the CAN hardware. In case that there is a problem in the CAN hardware initialization process, an error is indicated by the CPU LED, blinking 16 times, during the boot process. The controller then continues the boot process and can still communicate in RS232. If this error occurs, please contact Nanomotion experts for further instructions.
- **Hardware and Software Motion Limits:** The controller software continuously checks both the hardware and software limits. Whenever a limit is detected, any ongoing motion is stopped. Hardware limits are actual hardware signal lines. Software limits are low (and high) position values, beyond (and above) which the error is asserted. An FLS (Forward Hardware Limit) or High Software Limit stops positive motions only (towards increasing position value). An RLS (Reverse Hardware Limit) or Low Software Limit stops negative motions only (towards decreasing position value). During Limit Stop Condition, the controller uses the "DL" (Deceleration on Limit) value for the deceleration profile.
- **Torque Limit:** The torque limit protection is continuously monitoring the driver command value, and limits the maximal command. As noted above, the Driver command limitation has two different parameters, TL (which is the ultimate command saturation limit), and IS which can (separately from TL) limit the Integral value. This is needed in some cases to improve dynamic responses. It should be noted that the value of TL overrules the value of IS (see chapter 4, Part II, for further details about the

control filter structure). The TL saturation limit is operational is all enabled motor states (both open and closed loop modes).

5.4 Special Handling of Software Limits

In the FlexDC, when a Begin Motion command (BG) is issued in PTP mode (MM=0), beyond a software limit, the BG command fails with a new Error Code type: “EC=53”, “SW_LIMIT_ERROR”.

The new Error Code is generated during the BG command, and only in PTP motion mode. When a “SW_LIMIT_ERROR” is generated, the command does not start.

This behavior is different from previous implementations that checked for S/W limits only during motion.

6 Advanced Features

This chapter describes the following FlexDC advanced controller features:

- Data Recording.
- Advanced Encoder Interfaces - Compare Events.
- Advanced Encoder Interfaces - Capture Events.
- Auxiliary Analog Interfaces.
- Dynamic Error Mapping Correction.

6.1 Data Recording

Data recording is a very powerful feature of the FlexDC that allows the user to record internal controller variables, store them in local temporary arrays, and upload them to a host computer using either one of the controller's communication channels. The user can of course access the recorded buffers from within a script program if required.

Data recording significantly improves the control filter adjusent process (control parameters tuning), application debugging and monitoring, and troubleshooting.

The FlexDC has an outstanding Data Recording capabilities, including the following:

- Simultaneous recording of up to 8 internal controller variables.
- Up to a total of 15,000 data recording points! The user can select to record 8 vectors 1,875 sample points each, 1 vector 15,000 sample points, or any other combination.
- Selection of more then 40 internal variables for each recorded vector.
- More then 50 spare variables to select from, for future firmware usage, are already supported in the existing Data Recording interface.
- Fast sampling rate of up to 122 μSec per sample point (for all selected vectors). The FlexDC supports Data Recording at the servo-sampling rate of 8,192 Hz. The user can choose to collect data samples at a slower rate using the Recording Gap parameter (see below).

Optional advanced triggering options. This option is not supported by the standard firmware version of the controller. Please consult Nanomotion experts for more information.

In the next sections the operation of Data Recording in the FlexDC firmware is explained.

6.1.1 Operating Data Recording in the FlexDC

The FlexDC firmware code supports Data Recording using the following Keywords:

- Begin / Stop Data Recording command.
- Data Recording Configuration Parameters:
    - ♦ Select Recorded variables parameter.
    - ♦ Select Recording Length parameter.
    - ♦ Select Recording GAP parameter.
- Report Recording Status parameter.
- Data Recording Array.

Instead of using these parameters and commands directly the user can control the Data Recording features of the FlexDC through the Nanomotion Shell Application (GUI). With a few mouse clicks, the user can select the recorded variables, initiate recording process, and view the resulted graphs in the advanced Data Viewer application. Read further in this manual for more information about the Nanomotion Shell Application support for Data Recording.

However, from time to time the user may choose to directly use Data Recording low-level keywords (bypassing the GUI). This may be useful for example to initiate a data recording process from within a script program, in order to synchronize the Data Recording process with a machine sequence. The next sections fully describe the FlexDC firmware Data-Recording interfaces.

6.1.2 Data Recording Keywords

This section describes the Data Recording keywords of the FlexDC.

6.1.2.1 Begin / Stop Data Recording Command – BR

Using this Command, the user can start or stop the data recording process. The command only sets internal flags that start the real time recording process. The command does not check the validity of recorded vectors whatsoever; except for non-current on-going recording process. The command syntax is as follows:

XBR,<Optional Parameter>

where:

- **X** is an axis identifier. Since “BR” is a global function (not related to any axis), calling it with any axis identifier starts (or stops, according to the parameter) the recording process.
- Parameter <Optional>: The “BR” command can receive an optional parameter. When called without any parameter, i.e. “XBR”, the command starts the recording process.
- Parameter=1, “XBR,1”: Start a new recording process. This is identical to “XBR”.
- Parameter=0, “XBR,0”: Stops the current ongoing recording process. “RR” is reset to “0” immediately.

When a new recording starts, “RR” (Recording Status) is automatically set to the value of “RL”, the total required number of sample points. As the recording process continues, on each sample point the value of “RR” is decremented by “1”. When recording is complete, “RR” is “0”. Only then it is possible to upload the recorded data.

The “BR” (or “BR,1”) Begin Recording command checks only that “RR” is zero before enabling a new recording process. If “BR” is issued during an active recording (while “RR>0”) the command is rejected, and a “STILL_RECORDING” Error Code #16 is generated.

Note that the controller does not check if previous buffers were uploaded or not. Issuing a new Begin Recording command always overrides old data.

“BR,0” does not check any conditions, and always stops the data recording process.

6.1.2.2 Select Recording GAP Parameter – RG

See the “RG” keyword reference in chapter 7, Part II, for more information about upload data recording data delays in CAN bus operation.

6.1.2.3 RG Parameter

The recording GAP "RG" defines an integer number gap (in 122 μSec servo sample intervals) between each two consecutive recording sample points. "RG" is used to allow data sampling at a slower rate then the servo sample rate.

When "RG=1" the data sampling rate equals the servo sample rate of 8,192 points/sec. When "RG=2" recorded data is sampled every second servo sample, i.e. at a rate of 4,096 points/sec. "RG=8" results in data sample rate of 1,024 points/sec, and so on.

6.1.2.4 RG[2] – Recording Upload Delay

When uploading large data buffers in CAN bus, the FlexDC can generate high loads on the CAN bus network. Depending on the PC load and type of CAN board, on high buffers upload, some CAN messages can be lost. In order to avoid this problem, the FlexDC can add delays between CAN messages during data recording upload. The Delay is set by RG[2], and is given in servo sample time multipliers.

RG[2]=0 means no delay. RG[2]=1 means 1 sample time delay (this is 61 micro-sec on the 4M and 122 micro-sec on the 2M) and so on.

Usually, a delay of 3-5 samples is sufficient for most cases.

6.1.2.5 Select Recording Length Parameter – RL

"RL" defines the number of data points per sampled vector. This number defines the final size of each recorded vector. "RL" value can be up to 15,000 if only one vector is selected to be recorded, or up to 1,875 if all vectors (up to 8) are selected for recording. For example, when "RG=8", and "RL=1,875", each vector will be ~2 seconds long.

Note that the Nanomotion Shell Application automatically appends a time vector to any recording file.

6.1.2.6 Report Recording Status Parameter – RR

"RR" is a read only parameter, indicating the recording status. When a new recording starts, the value of "RR" is internally set to the value of "RL". It is being automatically decremented by "1" at each sample point (every "RG" servo sample times). When "RR=0" recording is complete.

6.1.2.7 Select Recorded Variables Parameter – RV

The FlexDC supports simultaneous data vectors to be recorded at the same time. The user can of course select to record less then this vector. FlexDC supports 8 simultaneous data vectors to be recorded at the same time.

The definition of each recorded vector contents (the link to an internal controller variable) is done using the "RV" parameter. Currently, the following internal controller variables (see Table 7) can be selected for data recording for each one of the recorded vectors:

Recorded Variable Description	Axis Related	Variable Keyword
None (empty vector)	---	---
Encoder Position	Yes	PS
Encoder Velocity	Yes	VL
Position Error	Yes	PE
Desired Position	Yes	DP
Controller Output	Yes	PO
Status Register	Yes	SR
Motion Status	Yes	MS
Analog Input	Yes	AI
Motor Fault	Yes	MF
Input Port	No	IP
Output Port	No	OP
Reserved	---	---

Table 7: Internal Controller Variables for Data Recording

Notes:

- *By selecting a NULL variable value (RV=0) for a specific vector, this vector is disabled (not recorded).*
- *It is required that enabled Recorded Vectors is orderly arranged. This means that after the first NULL RV, all following axes RV's should be "0".*
- *Most of the variables are axis-related variables. This means for example, that the user can select to record for each recorded vector the value of XPS, YPS, etc.*
- *See the "RV" parameter keyword reference in chapter 7, Part II, for specific details about all possible "RV" values.*

6.1.2.8 DA and AR Arrays in FlexDC

FleXDC has a data-recording array "DA" in size of 16,000 points, but the recording length is limited to 15,000 points only. The general-purpose array "AR" size is 1,000 points, and overlaps the DA array in its first 1,000 points.

This means that DA[1÷1000] == AR[1÷1000].

To avoid over-running the AR array when data recording is initiated, the data recording starts from DA[16,000] and ends at DA[1001] (depending on the RL).

This implementation allows special applications to define large AR arrays (by accessing "DA" at locations higher then 1,000).

In all normal applications, when using "AR" in its defined limits (i.e. [1÷1000]), no overlap occurs, even when the full data recording buffers are used.

6.1.3 Data Recording Support in Nanomotion Shell Application

The user can select the recorded variables, configure recording length, initiate recording process, and view the resulted graphs in our advanced Data Viewer application. Refer to "Part IV– Nanomotion Shell Application" for more information about the support for Data Recording.

6.2 Position Compare Events

Position compare events is a hardware-supported feature of the FlexDC encoder interface that provides the ability to generate accurate hardware pulses based on comparing the actual encoder position with pre-defined values. When a compare condition is satisfied, a hardware pulse is automatically generated by the FlexDC, and is directed to one of the digital outputs of the FlexDC.

The compare feature is implemented by the FlexDC encoder hardware interface, so the actual delay between the exact compare time to the generated pulse is very short (few cycles of the internal 66 MHz encoder interface module clock, in the current hardware version). This feature is useful in applications like printing and scanning, where external hardware should be synchronized with actual encoder location.

The FlexDC supports simultaneous compare events on both of its two encoders, independent from one another. The user can configure the hardware to redirect a generated event pulse to any one of the controller digital outputs. This way a user working with a dual axes system (X/Y stage for example), requiring to generate compare event pulses based on the X and Y encoders alternatively, can use only one digital output, and control the source of the pulse to be an X or Y encoder Compare Event by using a simple software configuration.

Note that the current hardware version of the FlexDC supports two of its eight digital outputs as Fast Outputs. The standard FlexDC digital outputs interface is isolated and buffered. While this is good for normal outputs, when fast synchronization pulses are required, a faster interface is needed. Thus, the FlexDC supports the first four digital outputs as Fast Outputs. Outputs configured as Fast Outputs are non-isolated, and are driven by a TTL buffer. The Fast Outputs use the same pins as do the normal outputs of the controller (DOut5 and DOut6).

As a standard, the FlexDC generates a single hardware pulse for each compare event. The user can control the pulse width with a few software configurable options. However, the controller can optionally support any special output pulse sequence. For this option it is suggested to contact Nanomotion technical support.

In general, the FlexDC supports two modes of Compare Events Generation:

- **Mode 0:** Fixed GAP (incremental), Distance < 16 Bit.
- **Mode 2:** 32 bit Arbitrary GAP location tables.

In order to operate the Position Compare feature, there are a few dedicated parameters and a command that controls its operation.

In the following sections the operation of each one of the supported Compare Function modes is explained.

Note:

- ***Mode 1:*** *Fixed GAP (incremental), Distance > 16 Bit. FlexDC current firmware version does not support Mode 1.*
- ***Mode 3:*** *32 bit Arbitrary GAP location tables using the FPGA RAM. FlexDC current firmware version does not support Mode 3.*

6.2.1 Mode 0: Fixed GAP (Incremental), Distance < 16 Bit

In this mode, the FlexDC is programmed with the desired start point - *PStart*, desired end point - *PEnd*, and desired incremental GAP - *Distance*. The first pulse is always generated at the exact Start Position - *PStart*. The hardware then automatically increments (or decrements, see explanation below) the next compare point by the *Distance* value, and so on, until the *PEnd* is reached.

The first pulse is thus generated at: *Position* = *PStart*, the second is generated at: *Position* = *PStart* + *Distance*, the next one will be at: *Position* = *PStart* + *Distance * 2,* etc. In general, the N^{th} pulse is generated at position: *Position* = *PStart* + *Distance * N,* where N=0 is the start point – *PStart.*

In this mode the compare pulses are fully generated by the hardware, so there is no limit to the max pulses frequency. Distances as low as one encoder count, at any encoder speed, are supported.

The value of *Distance* (the incremental GAP) is limited to 16 bit, i.e. +/- 32,767 (excluding 0). The sign of *Distance* controls direction of operation. Positive *Distance* value defines increasing encoder counter motion. Negative *Distance* value defines decreasing encoder counter motion (see notes below).

The compare pulse in this mode is automatically disabled by the real time controller firmware when the end point condition is met. This is when: *Position* > *PEnd* for *Distance* > *0*, and when: *Position* < *PEnd* for *Distance* < *0*.

Notes:

- *Although in this mode the hardware is responsible for the exact compare triggering, it is the controller real time software (firmware) that manages the end point monitoring (i.e. disabling the compare pulse output when PEnd is passed). As a result, although the actual pulse frequency is not limited, if the resulting pulse frequency is higher then the servo sampling rate (currently 8,192 Hz), additional pulses might be generated beyond location PEnd. In any case, all pulses are disabled no later then 122 μSec after PEnd is passed.*
- *As noted, the value of Distance is limited to +/- 32,767, excluding 0. Although the parameter itself is not range protected, the compare function enable command validates all parameters, and issues a dedicated Error Code if any of the parameters is out of range.*
- *The Compare function works correctly **ONLY** if the sign of Distance corresponds to the direction of motion, and to PStart and PEnd definitions. This means, that for Distance > 0 the user **MUST** specify PEnd > PStart, and the motion direction **MUST** be positive (i.e. from lower encoder count, to higher encoder count). For Distance < 0 the user **MUST** specify PEnd < PStart, and the motion direction **MUST** be negative (i.e. from higher encoder count, to lower encoder count).*
- *If the above conditions are not met, the compare pulses are generated in unexpected positions.*

6.2.2 Mode 2: 32 Bit Arbitrary Tables

Mode 2 allows the user to define an array of 32 bit position locations, to specify arbitrary compare locations. In Mode 2 the user fills in the desired compare locations to the general-purpose array “AR”.

In the FlexDC up to 1,000 compare points may be defined (currently limited by the size of the “AR” array).

The user then defines the *IStart* and *IEnd* indexes (index entries on the “AR” array), from which the compare locations are taken. The *Distance* parameter needs to be defined as +1 for positive motions, and –1 for negative motions.

Notes:

- *In this mode the controller real time firmware code is responsible for table points location increment. This implies a practical limitation on the position distance (in encoder count units) between each two consecutive table points, depending on the actual motion speed. The limitation requires that the resulting max arbitrary location compare pulse frequency is smaller then 8,192 Hz (in the current product firmware version).*
- *In any case (regardless of the motion direction), IEnd should be greater than IStart. The exact conditions tested before the mode is enabled are:*

0 < IStart < IEnd <= 10,000

- *Similarly to Mode 0, here the positions in the "AR" array **MUST** be defined in a strict ascending or strict descending order, and comply with the Distance (actually direction) definition, and the actual motion direction. If these conditions are not met, the compare pulses are generated in unpredicted unexpected positions.*
- *The "AR" array (used for location table definitions) is a non-axis related array. The size of the "AR" Array is [1 x 1,000].*
- *Although all axes can operate simultaneously and independent from one another, when working in Mode 2, all axes share the same "AR" array. The user should use separate "AR" areas for each axis if more than one is needed to be operated in this mode.*

6.2.3 Compare Function Parameters, Activation and Error Codes

The FlexDC uses a special array "PG" (abbreviation stands for "Pixel Generation Parameters") to control the Compare function operation, and a new activation command "PQ". This section describes the option defined by each parameter, and the command syntax.

6.2.3.1 The "PG" Array

The "PG" array elements control the operation of the compare function. "PG" is an axis related array, sized [2 x 8]. Each axis has 8 parameters controlling the compare operation as described in Table 8:

Array Element	Function	Description
PG[i][1]	Operation Mode	This parameter controls the compare function mode of operation: 1. PG[i][1]=0 : Defines Compare Mode 0. 2. PG[i][1]=2 : Defines Compare Mode 2.
PG[i][2]	Distance and direction	For Mode 0 this parameter defines the auto-increment distance. The parameter should be limited to +/-32,767, excluding 0. For Mode 2 this parameter should be +1 for positive motions (incrementing position motions), and –1 for negative motions (decrementing position motions).
PG[i][3]	Start Point	For Mode 0 this parameter defines the Start Position (PStart) in encoder counts for the compare function. The first compare pulse is always be at exactly that point. For Modes 2 this parameter defines the Start Index (IStart) in the "AR" compare position table, corresponding to the first compare point. The first compare point is at the encoder location defined by "AR[Istart]".
PG[i][4]	End Point	For Mode 0 this parameter defines the End Position (PEnd) in encoder counts for the compare function. Beyond this location the compare function is automatically disabled. For Mode 2 this parameter defines the End Index (IEnd) in the "AR" compare position table, corresponding to the last compare point. The last compare point is at the encoder location defined by "AR[IEnd]".
PG[i][5]	Pulse Width	This parameter defines the pulse width (ignoring PG[i][6]): 1. PG[i][5]=0 : Pulse Width = 1 clock of 15 nano/sec. 2. PG[i][5]=1 : Pulse Width = 1.92 μSec. 3. PG[i][5]=2 : Pulse Width = 3.84 μSec. 4. ……. 5. PG[i][5]= 255: Pulse Width = 489.6 μSec (Max Value)
PG[i][7]	Pulse Polarity	This parameter defines the compare pulse polarity mode. 1. PG[i][7]=0 : Defines Normal (Positive) Pulse. 2. PG[i][7]=1 : Defines Inverted (Negative) Pulse.
PG[i][8]	Not used	Should not be assigned to any value for future compatibility.

Table 8: "PG" Array - Compare Function Parameters Description

Notes:

- *In Table 8, (i) represents the selected axis.*
- *In Incremental Mode 0, since the hardware automatically increments the compare match register, the actual compare condition is valid for only 2 basic hardware clock cycles (66 MHz).*
- *In the FlexDC, the "PG[i][6]" parameter (Pulse Width Mode) is not used. In turn, the pulse width parameter "PG[i][5]" is used to set the required pulse width, in multiplications of 1.92 μSec intervals. This implementation provides a more flexible user interface for defining the compare pulse width.*
- *The max allowed value for "PG[i][5]" is 255. This results in a Pulse Width of 489.6 μSec.*

In the arbitrary table supported Mode 2, the controller real time software is responsible for updating the compare match registers. As a result, the compare pulse width may be longer then requested. The start point of the pulse is however always matches the exact compare point without any delay.

6.2.3.2 The "PQ" Command

The "PQ" command is an axis-related command, enabling or disabling the Compare function for a specific axis. The command requires a parameter indicating the requested operation. The command syntax is as follows:

XPQ,Parameter

where:

- **X** is an axis identifier.
- For the current FlexDC version the compare function is supported for both axes – X and Y.
- Parameter=0: Indicates immediate disable of compare for the specified axis. No conditions are checked expect a valid axis identifier.
- Parameter=1: Indicates start compare function for the specified axis. The command validates correct parameter ("PG") for the specific requested mode.

In a situation where one of the command's parameters is out of range, the command returns an error prompt: "?>" or generates a script "Run Time

Error" (if called from within a script macro program). The relevant Error Code flags ("EC" or "QC") is updated to reflect the error cause.

Notes:

- *The user should be aware that not all conditions for correct operation of the Compare Function could be validated during command initialization. For example, the minimal distance between each two consecutive points in the "AR" table (in Mode 2) cannot be tested as the limitation depends on the actual motion speed. It is the user's responsibility to specify correct parameters values for each operation mode. Refer to the specific mode description section defining operation limitations in each mode.*
- *The error codes generated by the "PQ" command are presented below.*

6.2.3.3 Dedicated Error Codes related to the Compare Function Operation

As explained in the previous section, in case that the "PQ" command fails to validate one of its parameters, the command returns an error prompt: "?>" or generates a script "Run Time Error" (if called from within a script macro program).

The relevant Error Code flags (“EC” or “QC”) are presented in Table 9:

Val	EC/QC Code Name	Error Description
34	EC_PARAM_OUT_OF_RANGE	The “PQ” command’s parameter is allowed to be “0” for disable or “1” for enable. Issuing a “PQ” command with a parameter out of that range will issue this error code.
38	EC_PARAM_EXPECTED	The “PQ” command must receive a parameter. If “PQ” is issued without a parameter this Error Code is returned.
60	MODE_PARAM_NOT_VALID	This error is issued by “PQ,1” if the requested Compare Mode defined by PG[i][1] is out of its range. In the current firmware version only Modes 0 and 2 are supported.
61	PULSE_MODE_PARAM_NOT_VALID	This error is issued by “PQ,1” if the Pulse Width Mode Parameter defined by PG[i][6] is out of its range. The allowed range for the Pulse Width Mode Parameter is: “0” or “1”.
62	PULSE_WIDTH_PARAM_NOT_VALID	This error is issued by “PQ,1” if the Pulse Width Parameter defined by PG[i][5] is out of its range. The allowed range for the Pulse Width Parameter is: “0” to “3”.
63	PULSE_POL_PARAM_NOT_VALID	This error is issued by “PQ,1” if the Pulse Polarity Parameter defined by PG[i][7] is out of its range. The allowed range for the Pulse Polarity Parameter is: “0” or “1”.
64	PD_PARAM_NOT_VALID	This error is issued by “PQ,1” if the Distance Parameter defined by PG[i][2] is out of its range. Out of range values for Distance are: 0 in all modes. Out of +/-32,767 range in Mode 0. Not equal +1 or –1 in Mode 2.
65	PS_PE_PARAM_NOT_VALID	This error is issued by “PQ,1” if the Start Point or End Point Parameters defined by PG[i][3] and PG[i][4] are not valid. These parameters are validated only in Mode 2 (see specific operation mode description for more details about limitations on PStart and PEnd.

Table 9: Error Codes Generated by the "PQ" Compare Function

6.2.4 Configuring Digital Outputs for the Compare Function

The FlexDC has general-purpose digital output pins. There are six un-committed general-purpose digital outputs in the FlexDC, which are called Dout6_Fast. When not assigned as position compare event outputs, digital output pins can be controlled by the "OP" (Output Port) parameter. Each hardware digital output pin reflects the state of the corresponding bit in the output word parameter "OP" (see the "OP" parameter keyword reference in chapter 7, Part II, for more details).

When configured as Position Compare Event output, the actual hardware digital output pins in are controlled by the compare function. If the compare function is enabled without any output pin being assigned to it, no pulses are generated (the pin reflects the relevant bit value of "OP").

When an output pin is assigned to a position compare event function, its state is controlled by the compare logic hardware, and is not affected by the digital output word "OP". In the FlexDC, only Dout5_Fast and Dout6_Fast can be assigned as compare outputs.

It should be noted that when an output is assigned to a compare event, only its physical logic level is affected. The value of "OP" is not changed, and does not reflect in this case the actual hardware pin state.

The next two sections define how to assign digital outputs to the compare function and how to support fast (TTL) electrical interface.

6.2.4.1 Assignment of a Digital Output to a Position Compare Event

The FlexDC Hardware supports assignment for any of its 8 actual (physical) digital output pins as standard outputs, or as a position compare function output.

The digital outputs are configured using the **IO_MODE_0** select word (currently assigned using the "XOM" parameter (see the "OM" keyword reference in chapter 7, Part II, for further information).

This is a 32-bit array word, defined as follows:

- IO_MODE_0 – XOM Keyword

IO_MODE_0 : Bits 31÷0				
31÷4	3	2	1	0
Not used	**OM**			

Table 10: A 32-bit Array Word

As noted above, in the FlexDC, only Dout5_Fast and Dout6_Fast can be assigned as compare outputs. Digital output can be assigned as follows, using a 3-bit configuration field **OM**, as shown in Table 10, and the bit description below:

- Bits (1:0) – Defines the Dout5_Fast Output source:

00 – Standard Output, controlled by “OP”

01 – Output from compare channel X

10 – Output from compare channel Y

11 – Currently unused, for future purposes

- Bits (3:2) – Defines the Dout6_Fast Output source:

00 – Standard Output, controlled by “OP”

01 – Output from compare channel X

10 – Output from compare channel Y

11 - Currently unused, for future purposes

6.2.5 Position Compare Events Examples

The following example demonstrates initialization of X axis compare, to generate pulses at a fixed gap (Mode 0), starting from location 10,000 counts to location 100,000 counts, every 40 encoder counts. The pulse is directed to Output #1. Motion from location 0 to location 150,000 counts at Speed=100,000 is then executed. The resulted pulse frequency is 100,000 counts/sec / 40 counts/pulse=2,500 pulse/sec. When motion is completed, the function is programmed to generate pulses in the opposite direction (when moving back to location 0). Only the necessary parameters are re-configured.

```
‘ Disable any active compare for X Axis
‘ ------------------------------------
XPQ,0
‘
‘ Configure Digital Output #1 to be assigned as an X Axis
‘ Compare Output (All other outputs are standard Outputs)
‘ -----------------------------------------------------
XOM=1        ‘  (DOut1 is X Compare)
‘
‘ Initialize X axis Motion Parameters and reset position
‘ -----------------------------------------------------
XAC=1000000;XDC=1000000;XDL=1000000
XSP=100000;XPS=0;XMO=1;XAP=150000
‘
‘ Initialize the X Compare Function
-----------------------------------
XPG1=0          ‘ Set Mode 0
XPG2=40         ‘ Set Compare Distance
XPG3=10000      ‘ Set Compare Start Position
XPG4=100000     ‘ Set Compare End Position
XPG5=2          ‘ Set Pulse Width (=3.9 μSec)
XPG7=0          ‘ Set Pulse Polarity to Normal (Positive)
XPQ,1           ‘ Activate X Compare Function
‘
‘ Start X motion, and wait for end of motion
-------------------------------------------
XBG
@while (XMS != 0)     ‘ Wait for End Of Motion
@endwhile
‘
‘ Initialize the Compare in the opposite direction
-------------------------------------------------
XPQ,0           ‘ Disable X Compare
XPG2=-40        ‘ Set Compare Distance Negative Direction!
XPG3=100000     ‘ Set Compare Start Position
XPG4=10000      ‘ Set Compare End Position
XPQ,1           ‘ Activate X Compare Function
‘
‘ Start Backward X motion towards 0 position
-------------------------------------------
XAP=0;XBG
```

6.3 Position Capture Events

Position Capture (Latching) events are a hardware-supported feature of the FlexDC encoder interface that provides the ability to latch the exact encoder position register based on an external or internal hardware pulse.

The FlexDC hardware Capture mechanism support two type of trigger pulse sources:

- Capture Position Based on an Encoder Index Pulse, and
- Capture Position Based on a Digital Input Pulse.

FlexDC hardware fully supports the encoder hardware interface. The Nanomotion Shell Application can capture positions (based on either Index or Inputs), at any encoder speed. There is no limitation on the motion velocity.

This feature is useful for finding the exact (1 count resolution) homing location when operated on the encoder index, and to synchronously latch multiple axes system locations when operated on digital inputs.

The FlexDC supports simultaneous capture on both of its two axes.

The user can configure the Compare Pulse Source for each encoder independently from other channels.

6.3.1 Capture Modes

When operated on the Index pulse, the Capture uses the internal Index signal to latch the position. In this mode each axis can capture the position based on its own Index pulse. When based on digital inputs, the user can select any one of the 10 digital input lines to be the Capture pulse source for any axis, without any limitation. The same digital input line can be used to synchronously Capture the location of both axes at once.

Although each one of the controller's digital inputs can be used as a Capture input, in the current hardware version only two digital inputs (DInp9, DInp10) are supported as fast TTL inputs. As normal inputs are optically isolated, using standard inputs for Capture introduces a delay of a few microseconds. Fast inputs are TTL based, eliminating any delays.

6.3.2 Operating the Position Capture and Relevant Keywords

The Capture function is independent from any other operation mode of the controller. The function of Position Capture is simple. The user only needs to set the Capture source signal configuration word, and the controller automatically captures positions whenever the Capture source pulse is detected. There is no special activation command for the Capture function, nor any special error codes related to it. The following dedicated Keywords are used to configure and work with the Capture function:

- XN: Capture Index counter.
- XC: Last Capture Position.
- YOM: Configure the Capture Signal source for all axes.

In the following sections the usage of these keywords is explained.

6.3.3 The Capture Events Counter – "XN"

Each time the hardware Captures (Latches) a new location, the total number of Capture events ("XN") is incremented by "1". The user can reset this variable to "0", and monitor its value to wait for a Capture event within a script program. This can be used for example to signal events to a host computer whenever a Capture event is sensed.

"XN" is an axis related parameter keyword. Each axis holds its own Capture index counter.

6.3.4 The Capture Location – "XC"

The last Captured location is stored by the controller firmware in the "XC" parameter for each axis independently (i.e.: XXC, YXC). The user should note that when "PS" is updated, the value of "XC" is meaningless.

The Capture feature implementation does not support hardware or software buffers. Whenever a Capture is detected, the last value of "XC" is overridden and lost.

As indicated above, "XC" is an axis related parameter keyword. Each axis holds its own Captured Position Location value.

6.3.5 Selection of Capture Source Pulse – “YOM”

The user can configure the Capture pulse source by modifying the **IO_MODE_1** register. This is (in the current firmware version) done using the “YOM” parameter (see the “OM” keyword reference in chapter 7, Part II, for further information). This is a 32-bit array word, defined as follows:

6.3.5.1 IO_MODE_1 – YOM Keyword

IO_MODE_1 : Bits 15 ÷ 0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	**-**	**-**	**Pol**	**Y Axis – Capture Source Select**				**-**	**-**	**-**	**Pol**	**X Axis – Capture Source Select**			

The order of Bits in Each Byte is identical for all axes. The Bit order in each Byte is described below:

- Bits [0 – 3]: selects the X Axis – Capture Source:

“0000” X Event source is Din1. (0)

“0001” X Event source is Din2. (1)

“0010” X Event source is Din3. (2)

“0011” X Event source is Din4. (3)

“0100” X Event source is Din5. (4)

“0101” X Event source is Din6. (5)

“0110” X Event source is Din7. (6)

“0111” X Event source is Din8. (7)

“1000” X Event source is Din9_Fast. (8)

“1001” X Event source is Din10_Fast. (9)

“1010” X Event source is Index X. (10)

“1011” X Event source is Index Y. (11)

- **Bit 4** selects the polarity of the X axis capture event:

Bit 4 = 0 Select Normal (Positive) Pulse Polarity.

Bit 4 = 1 Select Inverted (Negative) Pulse Polarity.

- **Bit [7 – 5]** – Reserved. Should be “0” for future compatibility.
- **Bits [11 – 8]** selects the Y Axis – Capture Source:

“0000” Y Event source is DIN1. (0)

“0001” Y Event source is Din2. (1)

“0010” Y Event source is Din3. (2)

“0011” Y Event source is Din4. (3)

“0100” Y Event source is Din5. (4)

“0101” Y Event source is Din6. (5)

“0110” Y Event source is Din7. (6)

“0111” Y Event source is Din8. (7)

“1000” Y Event source is Din9_Fast. (8)

“1001” Y Event source is Din10_Fast. (9)

“1010” Y Event source is Index Y. (10)

“1011” Y Event source is Index X. (11)

- **Bit 12** selects the polarity of the Y axis capture event:

Bit 12 = 0 , Select Normal (Positive) Pulse Polarity.

Bit 12 = 1 , Select Inverted (Negative) Pulse Polarity.

- **Bits 31 – 13** : Reserved. Should be “0” for future compatibility.

6.3.6 Configuring Fast Digital Inputs for the Capture Function

The FlexDC has two dedicated fast inputs (Din9_Fast and Din10_Fast). There is no special configuration required for Fast Digital Inputs in the FlexDC.

6.3.7 Position Capture Events Examples

6.3.7.1 Capture and CompareExample

The following example demonstrates usage of the Capture and Compare functions. The X axis is programmed to generate Compare pulses on fixed GAP. The pulses are directed to Fast Digital Output #5. It is assumed that Dout5_Fast is physically connected to Din9_Fast. Axes X and Y are then programmed to Capture their locations on each Compare pulse. The Captured X position should be identical to the desired Compare position. The Captures Y position reflects the Y axis location when X was commanded to generate the Compare pulse. The captured positions are then sent through the CAN bus to a host computer. The Compare GAP is programmed to 200 encoder counts, while motion is at 100,000 counts/sec. The resulted Compare frequency is 500 Hz.

This application can be used when an X/Y scan is made, and in order to know the exact planar location of the system on each compare pulse.

```
‘ Disable any active co
mpare for X Axis
‘ ------------------------------------
XPQ,0
‘
‘ Configure IO_MODE_0: DOut#5 assigned as X Compare
‘ Configure IO_MODE_1: X Y use DInp#1 as their Capture Source.
‘ ------------------------------------------------------------
XOM=1      ‘ Set IO_MODE_0
YOM=2056      ‘ Set IO_MODE_1 (X/Y Use DInp#9 for Capture)
‘
‘ Initialize X/Y axis Motion Parameters and reset position
‘ ------------------------------------------------------
BAC=1000000;BDC=1000000;BDL=1000000
BSP=100000;BPS=0;BMO=1;BAP=150000
‘
‘ Initialize the X Compare Function
----------------------------------
XPG1=0           ‘ Set Mode 0
XPG2=200         ‘ Set Compare Distance
XPG3=10000    ‘ Set Compare Start Position
XPG4=100000   ‘ Set Compare End Position
XPG5=2           ‘ Set Pulse Width (=3.9 μSec)
XPG7=0           ‘ Set Pulse Polarity to Normal (Positive)
XPQ,1                    ‘ Activate X Compare Function
‘
‘ Start X/Y motion, and enter a Loop to wait for the Compare
‘ Pulses. Pulses are counted and after 100 the loop ends.
------------------------------------------------------------
BXN=0;XICA=0
XZI1=3           ‘ Remote MSG sent to CAN Address = 3
BBG

#XCAPI1:
   @while (XXN == XICA) ‘ Wait for Next Event
   @endwhile
   @ XICA=XXN
   @ XICA=XICA+1           ‘ Increment counter
   BXC};XZM,2           ‘ Send Last Event

   @if (XICA > 100)       ‘ Check End Condition
      XJP,#XCAPIEND
   @endif

   XJP,#XCAPI1

#XCAPIEND:
   XZM,"END"

XQH                     ‘ Program Done.
```

Note that since X and Y Capture occurs simultaneously only the XXN is checked to detect the next event.

6.3.7.2 Latching the Index Location of an Axis

The next example demonstrates simple usage of the Capture mechanism to latch the Index location of the X axis. This can be combined in a simple Homing process to perform exact Index based homing process. This can be done at any motion speed. It is recommended to check that only One Index was found (usually in Rotary Motors), to avoid full motor revolution homing index error.

```
‘ Initialize X axis Motion Parameters and reset position
‘ ------------------------------------------------------
XAC=100000;XDC=100000;XDL=100000
XSP=10000;XPS=0;XMO=1;XAP=10000
‘
‘ Configure IO_MODE_1: Use X Axis Compare on Index
‘ ------------------------------------------------
YOM=10          ‘ Set IO_MODE_1 – X Compare on Index
‘
‘ Start X motion, and enter a Loop to wait for the Index
‘ Pulse
--------------------------------------------------------
XXN=0
XBG

@while (!XXN)   ‘ Wait for Next Index
@endwhile

‘ Index is found. Stop the motion. The Index location
‘ is stored in XXC. Stop the program.
‘ ---------------------------------------------------
XST
XQH
```

6.4 Auxiliary Analog Input Interfaces

The FlexDC has two general-purpose analog inputs.

Analog inputs are nominally ±10v and are converted using 12 bits A2D's in the FlexDC.

The analog input values, as can be reported by the "AI" parameter (XAI / YAI in the FlexDC). "AI" is the value of the analog input after deducting the Offset parameter "AS", see Figure 11.

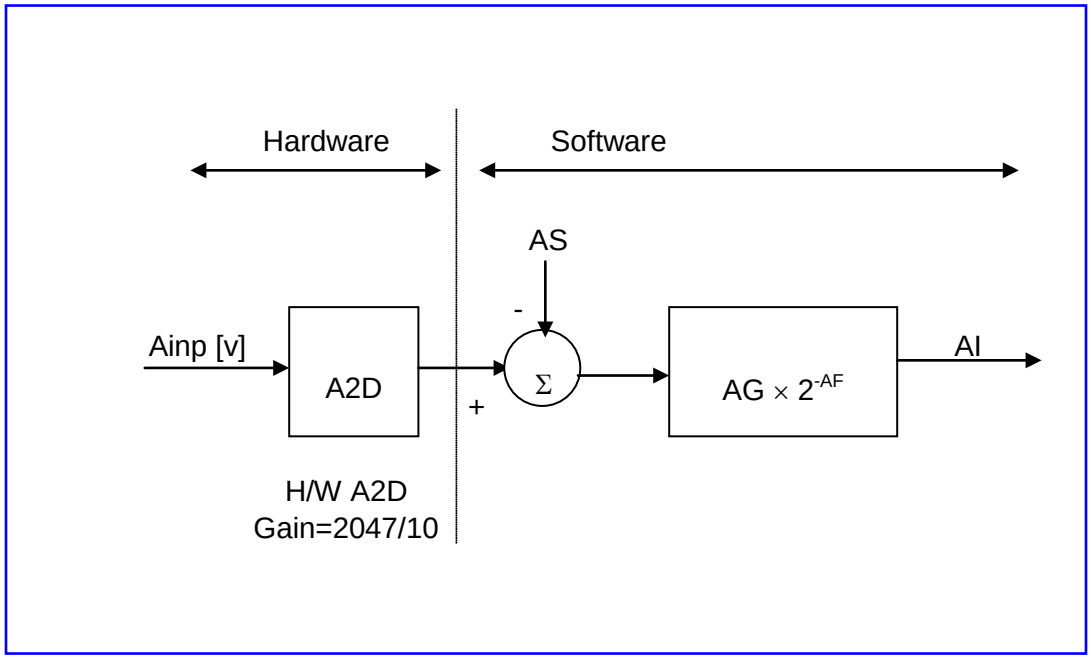


Figure 11: Analog Input Scaling Block Diagram

Note:

- *The analog inputs are sampled at ~ 1kHz (each input is sampled every 8 Servo cycles). For a complete description of the Analog Inputs Hardware circuits, refer to the "FlexDC User Manual".*

The analog input value is calculated and reported by the software variable "AI" according to the following equation:

$$AI = Floor\left[(Ainp \times A2DHWGain - AS) \times AG \times 2^{-AF}\right]$$

Notes:

- *Floor(x) truncates any non-integer value to an integer value towards minus infinity.*
- *Ainp × A2DHWGain is in the range of: -10v analog input result in nominal A2D reading of "0", 0v analog input result in nominal A2D reading of "2047" and a +10v analog input result in nominal A2D reading of "4095".*
- *AS, The Analog Offset parameter is in the range of: [0 ÷ 4095].*
- *Note that "AS" is decremented from the actual (positive) A2D reading value, so for example, in order to nominally achieved a symmetric AI reading, the value of "AS" should be +2047 and not –2047.*
- *The current implementation of "AI" computation formula dose not uses a dead-band function (although the dead-band parameter "AD" is supported, but has no effect).*
- *AG and AF parameters (the Analog Gain and Gain Offset) can be used to achieve any effective gain in the range of : ±219 (±524,288) to ±1/65,536.*
- *AG range is: ±219 (±524,288).*
- *AF range is: [0 ÷ 16], i.e. Gain Factor can be : [1/1 ÷ 1/65,536].*
- *The AG and AF parameters can be used to achieve very high or very low gains, or can combined together to achieve accurate floating point gains. For example, to achieve an overall gain of 4.125, use AG=33, and AF=3.*

Using the AG and AF parameters, the user can define any desired range for the AI value. For example, if: XAS=2047;XAG=100;XAF=2 and the analog input varies in the range of ±10 [v], Then:

XAI = ±10 * (2047/10) × (100 × 2^{-2}) = ±51,175

This is required for the Joystick motion modes. For example, the AI parameter is used as a speed reference for the Velocity Based Joystick Mode. Using "AG" and "AF", the AI value can be scaled to any desired velocity range.

"AS" can be used to compensate joystick or analog input circuits offsets. Note that nominally, "AS" should be 2047 to achieve "AI=0" for nominal 0v analog input value.

“AD” (the analog dead-band) is required for the Velocity Based Joystick Mode. Standard joysticks do not always return to the same zero value when they are released. This may cause a small velocity “drift” motion. “AD” can be used to define a range, at which the analog input is read as zero, avoiding any undesired motion.

In case a simple analog input reading is required, set parameters as follows: AD=0, AS=2047, AG=1 and AF=0. This provides a standard reading of ±2047 for an input of **approximately** ±10 [v].

6.5 Dynamic Error Mapping Correction

Dynamic Error Mapping Correction is required for the correction of non-linear mechanical position errors, caused for example by lead or ball screw. The correction is performed by interpolating desecrate positions user defined correction table, and altering the actual encoder position readings. Each axis can be corrected independently.

The correction table itself is defined in equally spaced intervals, between two maximum and minimum values of actual encoder readings. Beyond these values, the correction is fixed at the extreme table value point.

As a part of the real-time process, the true encoder position reading is corrected by a value that is taken from the correction table. When current position does not match an exact table point, linear interpolation is performed between two consecutive table points. Outside of table range, the last error correction value is used.

This option is not yet fully supported by standard firmware revisions. Please consult Nanomotion experts for more information.

7 Keywords Reference

This chapter describes the keywords supported by the FlexDC firmware. As discussed, the controller language defines two groups of keywords:

- Parameters Keywords.
- Command Keywords.

As noted there, each parameter owns a set of internal attribute flags defining the behavior of the Interpreter Module in response to each keyword received, such as if the Keyword is Axis Related or not, is the Keyword is a parameter or command, etc.

7.1 Keywords Attribute Reference

The following table describes the FlexDC Keywords Attributes List.

Note that some of the attributes are internal only, while some other are currently not used.

All internal and not used attributes are given for reference purpose only, and are designated in **GRAY** font. Attribute values are also used internally (by the controller Firmware), and are given for reference purpose only.

In the table below the abbreviation “KW” stands for “Keyword”. Where “Need” is used, this means that in order for the clause to be executed correctly, the condition defined there should be met. For example, the command “BG” (begins a new motion) needs of course its relevant motor to be “ON” (i.e. Enabled).

Attribute Definition	Attribute Value	Attribute Description
CPA_MOTOR_ON	0x00000001	Needs Motor ON
CPA_MOTOR_OFF	0x00000002	Needs Motor OFF
CPA_MOTION_ON	0x00000003	Needs Motion ON
CPA_MOTION_OFF	0x00000004	Needs Motion OFF
CPA_PARAM_IS_READ_ONLY	0x00000010	Parameter is Read Only
CPA_PARAM_IS_ARRAY	0x00000020	Parameter is Array
CPA_PARAM_SAVED_TO_FLASH	0x00000030	Parameter is saved to Flash Memory
CPA_PARAM_INIT_NEEDED	0x00000040	Parameter needs initialization (Internal).
CPA_PARAM_LEN_BIT_0	0x00000100	Not used
CPA_PARAM_LEN_BIT_1	0x00000200	Not used
CPA_PARAM_SPECIAL_REPORT	0x00000300	Parameter Has Special Report Function
CPA_PARAM_SPECIAL_ASSIGN	0x00000400	Parameter Has Special Assign Function
CPA_COMMAND_ALLOWS_PARAM	0x00001000	Commands Allows a Number Parameter
CPA_COMMAND_ALLOWS_STRING_PARAM	0x00002000	Commands Allows a string Parameter
CPA_COMMAND_SPARE_1	0x00003000	Not used
CPA_COMMAND_SPARE_2	0x00004000	Not used
CPA_KW_IS_COMMAND	0x00010000	Keyword is a Command Keyword
CPA_KW_IS_AXIS_RELATED	0x00020000	Keyword is Axis Related
CPA_KW_IS_VIRT_AXIS_RELATED	0x00030000	Keyword is Virtual Axis Related
CPA_KW_SPARE_1	0x00040000	Not used
CPA_KW_SOURCE_MUST_BE_MACRO	0x00100000	KW Source Must be from MACRO Only
CPA_KW_SOURCE_MUST_BE_COM	0x00200000	KW Source Must be from Comm. only
CPA_KW_SOURCE_MUST_BE_RS232	0x00300000	KW Source Must be from RS232 only
CPA_KW_SOURCE_MUST_BE_CAN_MAIN	0x00400000	KW Source Must be from Main CAN Ch.
CPA_KW_SOURCE_MUST_BE_CAN_AUX	0x01000000	KW Source Must be from Aux CAN Ch.
CPA_KW_SOURCE_MUST_BE_USB	0x02000000	KW Source Must be from USB Channel
CPA_KW_SOURCE_MUST_BE_LAN	0x03000000	KW Source Must be from LAN Channel
CPA_KW_SPARE_2	0x04000000	Not used
CPA_KW_ALL_MACRO_HALTED	0x10000000	KW Must have all programs halted
CPA_SPARE_2	0x20000000	Not used
CPA_SPARE_3	0x30000000	Not used
CPA_SPARE_4	0x40000000	Not used

Table 11: FlexDC Keywords Attributes and Restrictions

Each command and parameter can have one or more attributes from the table above. In addition, each parameter has a default value (when not loaded from Flash Memory or when Flash Memory value is not valid, as well as Minimum and Maximum limit values.

7.2 Command Keywords List

Table 12 describes alphabetical list of the FlexDC Commands Keywords.

Command Keyword	Axis Related	Description	Restrictions
AB	Yes	Immediately Abort any motion	None
BG	Yes	Begins a new Motion	Motor ON
BR	No	Start data recording process	Not Currently Recording
KR	Yes	Kill (stop) repetitive PTP motions	None
LD	No	Load all parameters from Flash Memory	All Macro Programs Stooped
OC	No	Clear an output Bit (set bit Low)	None
OS	No	Set an output Bit (set bit High)	None
PQ	Yes	Activate / Disables Compare Mode	None
QK	Yes	Kill all motions and Programs	
RS	No	S/W Reset Controller	Communication Only, All Motors are disabled, and programs are stopped
ST	Yes	Stop any motion	None
SV	No	Save all parameters from Flash Memory	All programs are stopped
UD	No	Upload Recording Data	None
VR	No	Get Firmware and FPGA Versions	None
ZA	Prg. Related	Remote Assign CAN message	From Program Only
ZC	Prg. Related	Remote CAN Command	From Program Only
ZR	Prg. Related	Remote Report Can Message	From Program Only
ZM	Prg. Related	Remote Send CAN Message (String/Number)	None

Table 12: Commands Keywords List

Note:

- *Table 12 DOES NOT include any script programming related commands. Refer to "Part III– FlexDC Macro Language" for further reference on Script Program related functions.*

7.3 Parameters Keywords List

Table 13 describes alphabetical list of all the FlexDC parameters.

Notes:

- *Table 13 DOES NOT include any script programming related parameters. Refer to "Part III– FlexDC Macro Language" for further reference on script programming related functions.*
- *All parameters are represented in signed long (32bit) format. Some parameters may be restricted to a positive only value.*
- *Grayed parameters are not operational in the current released firmware version.*

7.3.1 Parameters Keywords List

Key Word	Axis Related	Description	Restrictions	Saved to Flash	Read Only	Reset Val	Array Size	Assignment Range
AC	Yes	Acceleration Value [counts/s^2]	None	Yes	No	---	---	512÷ 120,000,000
AD	Yes	Analog Input Dead Band	None	Yes	No	10	---	0 ÷ 2,047
AF	Yes	Analog Input Gain Factor	None	Yes	No	0	---	0 ÷ 16
AG	Yes	Analog Input Gain	None	Yes	No	---	---	-524,288÷ 524,288
AI	Yes	Analog Input Value	None	Yes	Yes	---	---	± 2,147,000,000
AP	Yes	Next Absolute Position Target	None	No	No	0	---	± 2,147,000,000
AR	No	General Purpose Array	None	Yes	No	---	1x1,000	± 2,147,000,000
AS	Yes	Analog Input Offset.	None	Yes	No	2047	---	0 ÷ 4,095
CA	Yes	Special Control Parameters Array	None	Yes	No	---	2x16	See note [2]
CB	No	CAN Baud Rate Settings	None	Yes	No	1	---	1 ÷ 20

[2] The "CA" array controls advanced features of the controller real time servo loop. Although not restricted by the interpreter module (allows range is ±2,147,000,000), the specific limitations of each element in the array should be checked in the "A1" command reference and in chapter 4, Part II.

Key Word	Axis Related	Description	Restrictions	Saved to Flash	Read Only	Reset Val	Array Size	Assignment Range
CG	Yes	Axis Configuration	Motor OFF	Yes	No	---	---	0 ÷ 127
DA	No	Data Recording Array	None	No	No	---	1x15,000	± 2,147,000,000
DC	Yes	Deceleration Value [counts/s^2]	None	Yes	No	---	---	512÷ 120,000,000
DL	Yes	Limit Deceleration [counts/s^2]	None	Yes	No	---	---	512÷ 120,000,000
DO	Yes	DAC Analog Offset	None	Yes	No	0	---	± 32,767
DP	Yes	Desired Position	---	No	Yes	0	---	± 2,147,000,000
EC	No	Last Communication Error Code	None	No	Yes	0	---	0 ÷ 100
EM	Yes	Last End Of Motion Reason.	None	No	Yes	0	---	0 ÷ 8
ER	Yes	Max Position Error Limit	None	Yes	No	---	---	1 ÷ 8,000,000
FF	Yes	Acc and Vel Feed-Forward Gain	None	Yes	No	0	2 x 2	0 ÷ 65,536
HL	Yes	High Software Limit for Motions	None	Yes	No		---	± 2,147,000,000
IA	No	Indirect Access Index Array	None	Yes	No		1 x 50	± 2,147,000,000
IL	No	Set Input Port Bit Logic	None	Yes	No	---	---	0 ÷ 16,777,215
IP	No	Get Input Port	None	No	Yes	---	---	N/A
IS	Yes	Integral Saturation Limit	None	Yes	No	---	---	1 ÷ 32,767
KD	Yes	PIV Differential Gain	None	Yes	No		2 x 2	0 ÷ 2,147,000,000
KI	Yes	PIV Integral Gain	None	Yes	No		2 x 2	0 ÷ 2,147,000,000
KP	Yes	PIV Proportional Gain	None	Yes	No		2 x 2	0 ÷ 2,147,000,000
LL	Yes	Low Software Limit for Motions	None	Yes	No		---	± 2,147,000,000
ME	Yes	Master Encoder Axis Definition	None	Yes	No			0 ÷ 3
MF	Yes	Motor Fault Reason	None	No	Yes			0 ÷ 255
MM	Yes	Motion mode	Motion OFF	Yes	No	---	---	0 ÷ 8
MO	Yes	Motor ON (Enable/Disable)	None	No	No	0	---	0 ÷ 1
MS	Yes	Motion Status	None	No	Yes			0 ÷ 8
NC	Yes	No Control (Enable open loop)	Motor OFF	No	No	0	---	0 ÷ 3
OL	No	Set Output Port Bit Logic	None	Yes	No	---	---	0 ÷ 255
OM	Yes	I/O Hardware Configuration [3]	None	Yes	No	---		- 2,147,483,648 ÷ +2,147,483,647
OP	No	Set/Get Output Port	None	No	No	---	---	0 ÷ 255

[3] The “OM” parameters are bit filled commands. Refer to the “OM” command reference for more information.

Key Word	Axis Related	Description	Restrictions	Saved to Flash	Read Only	Reset Val	Array Size	Assignment Range
PA	Yes	General Purpose Parameter Array	None	Yes	No		2 x 100	± 2,147,000,000
PE	Yes	Position Error	---	No	Yes	0	---	± 8,000,000
PG	Yes	Compare Function Parameters	None	Yes	No	0	2 x 8	See note [4]
PO	Yes	Control Drive Command	None	No	Yes		---	± 32,767
PS	Yes	Encoder Position Value	None	No	No	0	---	± 2,147,000,000
RA	No	Receiving CAN Address	None	Yes	No	---	---	0 ÷ 2047
RG	No	Recording Gap [5]	None	Yes	No	---	1 x 2	1 ÷ 16,384
RL	No	Recording Length [6]	None	Yes	No	---	---	1 ÷ 100,000
RP	Yes	Next Relative Position Target	None	No	No	0	---	± 2,147,000,000
RR	No	Recording Status	None	No	Yes	0	---	0 ÷ 100,000
RV	Yes	Recorded Variables	None	Yes	No	---	---	0 ÷ 211
SM	Yes	Special motion mode	No Motion	Yes	No	---	---	0 ÷ 8
SP	Yes	Speed (For Profiler Motions)	None	Yes	No	---	---	± 30,000,000
SR	Yes	Status Register	None	No	Yes			0 ÷ 8,388,607
TA	No	Transmitting CAN Address	None	Yes	No	---	---	0 ÷ 2047
TC	Yes	Torque (open loop) Command	None	No	No	0	---	± 32,767
TD	Yes	32 Bit Timer Down Parameter	None	No	No			0 ÷ 100,000,000
TL	Yes	Torque Limit	None	Yes	No	---	---	0 ÷ 32,767
TR	Yes	Target Radius	None	Yes	No			0 ÷ 32,767
TT	Yes	Target Time	None	Yes	No			0 ÷ 32,767
VA	No	Vector Acceleration	None	Yes				0 ÷ 100,000,000
VD	No	Vector Deceleration	None	Yes				0 ÷ 100,000,000
VL	Yes	Actual Velocity	None	No	Yes			± 30,000,000
VS	Yes	Vector Speed	None	Yes	No			± 30,000,000

[4] The “PG” array element’s range is restricted by the “PQ” command depending on the compare function operation mode. Refer to the relevant command’s references (“PG”, “PQ”) and the “Advanced Features” section about the compare feature in this user manual.

[5] The Recording Gap parameter (“RG”) is now a [1 x 2] array. “RG” or “RG[1]” is the recording Gap. “RG[2]” defines a delay for upload Recording data buffers in CAN bus mode only. Please see the “RG” command reference for more information.

[6] The “RL” Recording buffer Length defines the number of max recorded data points per vector. It can be 15,000 points for one vector, or 1,750 for 8 vectors (and anything in-between). See the “RL” command reference and the section “Data Recording” in this User Manual for more information.

Key Word	Axis Related	Description	Restrictions	Saved to Flash	Read Only	Reset Val	Array Size	Assignment Range
WT	Yes	Wait time for Repetitive PTP	None	Yes	No	---	---	0 ÷ 800,000,000
WW	Yes	Smoothing Factor	None	Yes	No			0 ÷ 12
XC	Yes	Last Capture (Latch) Pos Value	None	No	Yes			± 2,147,000,000
XN	Yes	Number of Capture Events	None	No	No			0 – Only

Table 13: Parameters Keywords List

7.4 Keywords List – Functional Groups

The following section describes the FlexDC Keywords list ordered in functional groups.

7.4.1 Keywords Group Description

The following Keyword Groups are distinguished:

- Motion and Profiler Related Keywords.
- Control Filter and Real time Servo Loop Keywords.
- Data Recording Related Keywords.
- Special Features Interface Function Keywords.
- I/O Function Keywords.
- Script Programming Keywords.
- Configuration and Protection Keywords.
- General Keywords.

7.4.2 Keywords Groups

The following list describes all the FlexDC keywords (excluding Script Programming Keywords) divided to the logical groups indicated above.

7.4.2.1 Motion and Profiler Related Keywords

Keyword	Description
AB	Abort Command – Immediately stops any motion.
AC	Acceleration value in [counts / sec^2] for all Profiler based motions.
AP	Next Absolute Position for PTP Motions.
BG	Begins a new Motion Command.
DC	Deceleration value in [counts / sec^2] for all Profiler based motions.
DL	Limit Deceleration value in [counts / sec^2] for all Profiler based motions.
EM	Last End of Motion Reason.
FR	Gearing Mode Following Ratio
KR	Kill (stop) repetitive PTP motions
ME	Master Encoder Definition for Gearing Motion Mode.
MM	Defined the next Motion Mode, e.g.: PTP, JOG, etc.
MS	Motion Status Definition
RP	Next Relative Position for PTP Motions.
SM	Defines Special motion modes (Repetitive, etc).
SP	Defines Cruise Speed in [counts / sec] for all Profiler based motions.
ST	Stop Motion Command
WT	Defines delay (in units of $^1/_{16384}$ sec) for repetitive PTP motions.
WW	Profile Smooth Factor parameter
VA	Vector Acceleration (for XY Vector Motions)
VD	Vector Deceleration (for XY Vector Motions)
VL	Vector Limit Deceleration (for XY Vector Motions)
VS	Vector Speed (for XY Vector Motions)

Table 14: Motion and Profiler Related Keywords

7.4.2.2 Control Filter and Real Time Servo Loop Keywords

Keyword	Description
CA	Special Control Parameters Array.
DP	Desired Position. Holds the actual Position Reference.
ER	Max allowed Position Error.
FF	Acceleration and Velocity Feed-Forward Gains
KD	Control Filter Diff Term Gain
KI	Control Filter Integral Term Gain
KP	Control Filter Proportional Term Gain
PE	Actual servo loop Position Error.
PO	The Control Drive Command
PS	Position. Holds the actual encoder position value.
IS	Integral Term Saturation of PIV control filter.
SR	Status Register
MO	Motor ON – Enables (MO=1) / Disables (MO=0) the servo loop.
NC	No Control – Enables (NC=1) / Disables (NC=0) open loop mode.
TC	Torque Command in open loop mode.
TL	Torque Limit – Limits the D2A command – All modes.
TR	Target Radius
TT	Target Time

Table 15: Control Filter and Real time Servo Loop Related Keywords

7.4.2.3 Data Recording Related Keywords

Keyword	Description
BR	Begin Data Recording.
DA	Data Recording Array – size 1 x 100,000.
RG	Set Recording GAP (in units of $^{1}/_{16384}$ sec).
RL	Set Recording length (buffer length).
RR	Report Recording Status.
RV	Set the recorder variables.

Table 16: Data Recording Related Keywords

7.4.2.4 Special Features Interface Function Keywords

Keyword	Description
AR	General purpose Array – size 1 x 10,000. This array is also used for 32 bit locations table definitions in Mode 2 of the Position Compare Events Function.
OM	Set I/O Modes Hardware Configuration. This keyword is used to configure the Compare and Capture functions. See also I/O functions Group.
PG	Compare Function Parameters Array – size 10 x 8. This array defines the parameters for the Position Compare Events Function operation.
PQ	Enable / Disable Position Compare Events Function Command for a specific axis.
XC	Capture Location. The “XC” parameter holds the last captured position of an axis.
XN	Capture Events Counter. This parameter is automatically incremented by the firmware on each Capture Event.

Table 17: Special Encoder Interface Related Keywords

7.4.2.5 Analog and Digital I/O Function Keywords

Keyword	Description
AD	Analog Dead Band
AF	Analog Input Gain Factor
AG	Analog Input Gain
AI	Analog Input
AS	Analog Input Offset
AO	Auxiliary Analog Outputs Command
DO	Analog Output DAC Offset
IL	Input Logic Bit Array.
IP	Input Port.
OC	Output Clear Bit.
OL	Output Logic Bit Array.
OM	Set I/O Modes Hardware Configuration. This keyword is used to configure the Compare and Capture functions. See also Table 17: Special Encoder Interface Related Keywords.
OP	Output Port.
OS	Output Set Bit.

Table 18: I/O Functions Related Keywords

7.4.2.6 Communication and Configuration Keywords

Keyword	Description
CB	CAN Bus – Baud Rate.
RA	CAN Bus – Receiving CAN Address.
TA	CAN Bus – Transmitting CAN Address.
CG	Specific Axis Configuration.
EC	Last Communication Error Code.
QC	Last Program Error Code.

Table 19: Communication and Configuration Keywords

7.4.2.7 Protection Keywords

Keyword	Description
IS	Integral Term Saturation of control filter (see chapter 4, Part II)
TL	Torque Limit – Limits the D2A command – All modes.
LL	Low Software Limit
HL	High Software Limit
MF	Motor Fault Reason Report

Table 20: Protection Keywords

7.4.2.8 General Keywords

Keyword	Description
AR	General purpose Array – size 1 x 10,000.
DA	Data recording Array – size 1 x 100,000.
IA	Indirect Access – General Purpose Array.
PA	General Purpose Parameters Array
LD/SV	Load from and Save to Flash Memory (Parameters and Script Program)
RS	Software Reset Controller Command
VR	Get Firmware Version Command

Table 21: General Purpose Related Keywords

7.4.2.9 Programming Keywords

The FlexDC servo controllers have a powerful script engine that allows running up to two (FlexDC) programs simultaneously, at very fast rates.

Combined with our Integrated Script Development and Debugging Environment (IDE), the FlexDC motion controller's internal programming engine provides endless capabilities for user application development, starting from simple homing routines, up to full machine sequences management. Refer to "Part III– FlexDC Macro Language".

7.5 Keywords List

The following section presents the FlexDC Keywords list (excluding Script Programming Keywords) in alphabetical order, including detailed definitions of each command and examples.

The description of each keyword includes:

- Purpose: The operation or task of the keyword.
- Attributes: See below.
- Syntax: Valid clause syntax.
- Example: Simple example of the keyword usage.
- See also: Related commands.

The following list describes all the valid keyword **Attributes**:

- Type: Command / Parameter.
- Axis related[7]: Yes / No.
- Array[8]: Yes (dimension) / No.
- Assignment[9]: Yes / No (i.e. Read Only).
- Command Allows Parameter[10]: Yes (Number / String / Both) / No.
- Scope: Communication / Program / Both
- Restrictions: See below.
- Save to Flash: Yes / No.
- Default Value: Yes (value) / No.
- Range: Min ÷ Max.

The following list describes all the valid keyword **Restrictions:**

- None.
- Keyword Needs No Motion.
- Keyword Needs Motion.
- Keyword Needs Motor OFF.
- Keyword Needs Motor ON.

[7] Axis or related (keyword's preceding character X or Y affects the keyword behavior).
[8] Applicable for parameters only.
[9] Applicable for parameters only.
[10] Applicable for commands only.

7.6 AB – Abort Motion Command

Purpose:

The “AB” Abort command aborts any motion immediately, without any profile. The motion is stopped abruptly in the next servo interrupt following the Abort command.

The “AB” command should be used in emergency cases only. Normally, the “ST” or “KR” commands should be used to stop any type of motion. Note that if an Abort command is issued when a motor is moving at high speed, the servo loop may be disabled due to high error.

Attributes:	**Type:**	Command.
	Axis related:	Yes.
	Array:	---.
	Assignment:	---.
	Command Allows Parameter:	No.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

```
XAB;          ‘ Aborts X Motion
AAB           ‘ Abort motion of All axes.
```

Examples:

The following code example shows starting a normal motion in X axis from Position “0” to Position “100,000”, and then aborting the motion.

```
XMO=1;XPS=0          ‘ Enables the motor and Set Position = “0”.
XMM=0;XSM=0          ‘ Set Normal Point To Point Motion Mode.
XAP=100000           ‘ Set Next PTP absolute location to “100,000” counts.
XAC=90000;XDC=90000  ‘ Set AC=DC=90,000
XSP=25000            ‘ Set Speed to “25,000”.
XBG                  ‘ Start a Motion
XAB                  ‘ Immediately aborts the X motion.
```

See also:

BG, ST, KR, ER

7.7 AC – Acceleration

Purpose:

To outline the normal Acceleration value to cruise velocity in all motion modes (that use the internal Profiler). This value is used to set the motion profile acceleration value in PTP, JOG, Motion modes, etc. The Acceleration value is defined in units of: [counts/sec^2].

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	100,000.
	Range:	512 ÷ 120,000,000.

Syntax:

```
XAC=1000000;     ‘ Set X Axis AC=1,000,000.
AAC=240000       ‘ Set AC=250,000 all axes.
```

Examples:

The following code example shows starting a normal motion in X axis from Position “0” to Position “100,000”, using Speed and Acceleration values.

```
XMO=1;XPS=0      ‘ Enables the motor and Set Position = “0”.
XMM=0;XSM=0      ‘ Set Normal Point To Point Motion Mode.
XAP=100000       ‘ Set Next PTP absolute location to “100,000” counts.
XAC=250000       ‘ Set Acceleration to “250,000”.
XDC=500000       ‘ Set Acceleration to “500,000”.
XSP=25000        ‘ Set Speed to “25,000”.
XBG              ‘ Start a Motion
```

See also:

DC, DL, SP, BG

7.8 AD – Analog Input Dead Band

Purpose:

Set the Analog Input Dead Band range.

See the “AI” (Analog Input) command reference for complete information about Analog Input interfaces support.

Note:

Current firmware revision does not support dead band in the analog input interface. Although the “AD” parameter is fully supported by the communication interface, it has no other effect. Analog Input value always assumes “AD=0”.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	10.
	Range:	0 ÷ 2,407.

Syntax:

```
XAD=10;     ‘ Set X Axis AD=10 (10 LSB of the Analog Input).
AAD=0       ‘ Set AD=0 to all axes (No Dead Band).
```

Examples:

See Syntax above.

See also:

AF, AG, AI, AS

7.9 AF – Analog Input Gain Factor

Purpose:

Set the Analog Input Gain Factor Multiplier.

See the “AI” (Analog Input) command reference for complete information about Analog Input interfaces support.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	0 ÷ 16.

Syntax:

```
XAF=2;      ‘ Set X Axis AF=2 (Gain factor is ¼).
AAF=0       ‘ Set AF=0 to all axes (Gain Factor 1:1).
```

Examples:

See Syntax above.

See also:

AD, AG, AI, AS

7.10 AG – Analog Input Gain

Purpose:

Set the Analog Input Gain.

See the “AI” (Analog Input) command reference for complete information about Analog Input interfaces support.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	-524,288 ÷ 524,288.

Syntax:

```
XAG=10;        ‘ Set X Axis AG=10.
AAF=1          ‘ Set AG=1 to all axes.
```

Examples:

See syntax above.

See also:

AD, AF, AI, AS

7.11 AI – Analog Input

Purpose:

Report the analog input value. The analog input value is calculated and reported by the software variable “AI” according to the following equation:

$$AI = Floor\left[(Ainp \times A2DHWGain - AS) \times AG \times 2^{-AF}\right]$$

“AS” The Analog Offset parameter is in the range of: [0 ÷ 4095]. Nominal value of AS=2047 results in a nominal “AI” reading of “0”. Using the “AG” and “AF” parameters for scaling, the user can define any practical desired range for the AI reading value.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	No.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

AAI

7.12 AP – Absolute Position

Purpose:

Defines the Next Motion Absolute Position (in counts) target.

The absolute position value is used by the controller as the next target position in both the PTP and Repetitive PTP motion modes. Upon a “BG” (begin motion) command, the controller generates a profile from the current desired (“DP”) position to the current “AP”. Note that in relative motion, the “RP” command simply changes the value of the “AP”.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows parameter:	---
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	-2,147,000,000÷2,147,000,000.

Syntax:

```
XAP=100000;          ‘ Set X Axis Absolute Position to “100,000”.
AAP=0          ‘ Set AP=0 in all axes.
```

Examples:

The following example shows resetting the X axis position to “0’, and then initiate a normal motion in X axis from Position “0” to Absolute Position “100,000”.

```
XMO=1          ‘ Enables the X Motor
XPS=0          ‘ Set X axis encoder Position = “0”.
XMM=0;XSM=0    ‘ Set Normal Point To Point Motion Mode.
XAP=100000     ‘ Set Next PTP absolute location to “100,000” counts.
XAC=250000     ‘ Set Acceleration to “250,000”.
XDC=500000     ‘ Set Acceleration to “500,000”.
XSP=25000      ‘ Set Speed to “25,000”.
XBG            ‘ Start a Motion
```

See also:

DP, RP, PS, BG

7.13 AR – General Purpose Array

Purpose:

"AR" is a user general-purpose array. The "AR" array is a non-axis related array, with a size of 1,000 elements. Each element in the array is a LONG format number, which can be assigned, with any value at any time.

Currently, "AR" is also used internally by the Compare mechanism, to define user 32 bit tables for the compare mode. For further information see section 6.2.2.

The index range of the "AR" array is: 1 ÷ 1,000. Since "AR" is non-axis related, accessing XAR, YAR, BAR, etc. actually access the same array element.

Refer to the "DA" array for further information regarding the "AR" parameter.

Attributes:	**Type:**	Parameter.
	Axis related:	No.
	Array:	Yes, size = [1][1,000].
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	- 2,147,000,000 ÷ 2,147,000,000.

Syntax:

```
XAR[1]=0;          ‘ Set AR[1] “0”.
AAR[300]=1000      ‘ Set AR[300]=1,000.
```

See also: Compare Functions.

7.14 AS – Analog Input Offset

Purpose:

Set the Analog Input Offset.

See the “AI” (Analog Input) command reference for complete information about Analog Input interfaces support.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	2,047.
	Range:	0 ÷ 4,095.

Syntax:

```
XAS=0;          ‘ Set X Axis AS=0 (no offset).
AAS=2047              ‘ Set AG=2047 to all axes.
```

Examples:

See Syntax above.

See also:

AD, AF, AG, AI

7.15 BG – Begins a New Motion Command

Purpose:

The “BG” command begins a new motion, according to the current motion mode. The “BG” command allows receiving an argument (parameters). The parameter may be omitted to start a normal single axis motion, or (currently in this version), be used (“-1”) to initiate a common “X/Y” vector motion.

In the FlexDC, if the motion mode is Point-To-Point (MM=0), and the motion is in to one of the software limits, the “BG” shall return an error, and the “EC” (if command performed via communication) or “QC” (if command performed via macro) shall be set to ErrorCode , EC=53.

Attributes:	**Type:**	Command.
	Axis related:	Yes.
	Array:	---.
	Assignment:	---.
	Command Allows Parameter:	Yes, Number (-1).
	Scope:	All.
	Restrictions:	Needs Motor ON and No Motion.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

```
XBG;        ‘ Start X Motion
BBG         ‘ Start motion in X and Y (non-synchronized).
BBG,-1      ‘ Start vector X/Y motion..
ABG         ‘ Start Motion in all axes.
```

Examples:

The following code example shows starting a normal motion in X axis from Position “0” to Position “100,000”, and waiting for end of motion.

The example can be written as a script program file. The main routine name is “#MOVX”, and can be executed and tested.

```
‘
‘ Routine to Move to Position 100,000 and wait for end of motion.
‘ ------------------------------------------------------------------------
#MOVX

XMO=1;XPS=0              ‘ Enables the motor and Set Position = “0”.
XMM=0;XSM=0              ‘ Set Normal Point To Point Motion Mode.
XAP=100000               ‘ Set Next PTP absolute location to “100,000” counts.
XAC=90000;XDC=90000      ‘ Set AC=DC=90,000
XSP=25000                ‘ Set Speed to “25,000”.
XBG                      ‘ Start a Motion
‘
‘ Wait for End Of Motion
‘ ----------------------------
@while (XMS != 0)        ‘ Wait for MS (Motion Status) top be “0”.
@endwhile
‘
XQH                      ‘ Stop program execution.
```

See also:

ST, KR, AB, MM, MS, and VA, VD, VL, VS about Vector Motions.

7.16 BR – Begin Recording Command

Purpose:

The “BR” command begins a new data recording sequence. The “BR” command assumes that the recorded variables and parameters are configured.

The “BR” command allows receiving an argument (parameter). “XBR” and “XBR,1” both start a new recording sequence. “XBR,0” terminates the current data recording process.

The “BR” (or “BR,1”) command checks whether the last recording session was terminated, and issues a “STILL_RECORDING” Error Code #16 if not (i.e. if RR>0). Data Recording can be started only when previous recording session was terminated. Note that the controller does not check if previous buffers were uploaded or not. Issuing a Begin Recording command always overrides old data.

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Command Allows Parameter:	Yes, Number (0, or 1).
	Scope:	All.
	Restrictions:	BR or BR,1 - Needs recording OFF.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

```
XBR      ‘ Start Data recording.
XBR,1    ‘ Start Data recording.
XBR,0    ‘ Stop Data recording.
```

See also:

RG, DA, RL, RR, RV

7.17 CA – Special Control Parameters Array

Purpose:

"CA" is a special control parameters array, allowing the user to further configure the servo loop features. "CA" is an axis related array, valid for axes X,Y. The size of the "CA" array is [2]x[16]. Each parameter in the CA Array controls a certain feature as explained below ('i' indicates an axis identifier for X and Y).

Array Element	Function	Description
CA[i][1 ÷3]	Not used	Should be "0" for future compatibility
CA[i][4]		This parameters defines the 2nd PIV filter duration. To disable the 2nd PIV set: CA[i][4]=0. The duration is defined in servo-samples units. I.e.: Value of 1 is 1/16384 of a second. Value of 164 is 10 mili-sec. The recommended value range for the 2nd PIV filter duration is: 0 <= CA[i][4] <= 16384. Negative values should be avoided.
CA[i][5,6]	Not used	Should be "0" for future compatibility
CA[i][7 ÷9]	2nd order filter parameters	These 3 parameters control the servo-loop 2nd order filter operation: Filter Gain, Filter Bandwidth and Q factor (or damping ξ).
CA[i][10 ÷12]	Not used	Should be "0" for future compatibility
CA[i][13]	2nd order filter Enable bit	This parameter Disables (if "0") or Enables (if !=0) the servo-loop 2nd order filter operation. For future compatibility, the value of CA[i][13] should only be set "0" for Disable, and "1" for Enable.
CA[i][14 ÷16]	Not used	Should be "0" for future compatibility

Note About CA Parameters Range:

The "CA" array is not range checked by the communication interface. This means that any valid number in the 32 bit range (± 2,147,000,000) can be set to any of the "CA" parameters. This however should be carefully avoided Users **MUST** comply to the parameters range setting as defined in the table above for each specific parameter!

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	**CA** [2]x[16]
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	See note above

Syntax:

```
XCA[13]=1;          ' Enable 2nd order filter for X Axis".
ACA[4]=0            ' Set CA[4]=0 for all axes (Disable 2nd PIV).
```

Examples:

The following commands enable 2nd order filter operation for the X Axis with the following parameters: *f=100 Hz, ξ=0.7*

```
XCA[7]=1537305 ; XCA[8]=127552 ; XCA[9]= -62110    ' Set Filter Parameters
XCA[13]=1                                          ' Enable X 2nd order filter
```

See also:

KP, KI, KD

7.18 CB – CAN Baud Rate

Purpose:

To set the CAN baud-rate. The CAN baud rate must be saved to the Flash Memory, and the controller must be reseated in order to change the CAN baud rate. See the notes about Initialization of CAN bus parameters in the RA and TA command references.

Currently the following baud rates are supported:

CB = 1 : Can Baud = 1 Mbps.
CB = 2 : CAN Baud = 500 kBps
CB = 4 : CAN Baud = 250 kBps
CB = 8 : CAN Baud = 125 kBps
CB = 10 : CAN Baud = 100 kBps
CB = 20 : CAN Baud = 50 kBps

Attributes:		
	Type:	Parameter.
	Axis related:	No.
	Array :	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	1.
	Range:	1 ÷ 20.

Syntax:

```
XCB=1        ‘ Set CAN Baud rate to 1 Mbps.
XCB          ‘ Report value of CB.
```

See also:

RA, TA

7.19 CG – Axis Configuration

CG – Axis Configuration for the Controller

Purpose:

"CG" is an axis-related parameter, defining specific axis configuration. "CG" Currently supports 16 configuration bits (Bit #0 to bit #15) as described in the following table:

CG Bit (Zero Based)	Function	Description
0	Invert Main Servo Driver Command	This bit controls the MAIN Servo Driver command polarity (main servo driver analog or PWM command output: TC). When set to "0" the default polarity is invert, i.e. TC=+32767 results in an analog command voltage of –10v. When set to "1" the default polarity is non-invert, i.e. TC=+32767 results in an analog command voltage of +10v.
1	Invert Main Encoder	This bit controls the encoder polarity. Users can set or clear this bit to change the encoder reading direction. When set to "0" the default polarity is non-invert. When set to "1" the default polarity is invert.
2	Reserved	Should be left 0.
3	PIV	Must be set ONLY to "0" for PIV Control (Factory default).
4	Reserved	Should be left 0.
5	Enable Encoder Error Detection	This bit Disables (when set to "0") or Enables (when set to "1") the Hardware Encoder Error detection feature. Note that when enabled, the controller forces Driver Fault condition when encoder error is detected. This option must be used with encoders having (electrical) differential interface only. When single ended encoders are used, this bit must be disabled.
6	Reserved	Should be left "0".
7	Use Auxiliary Encoder Feedback	Must be set ONLY to "0" for Single Loop (Factory default).
8	Invert Auxiliary Encoder	This bit controls the Auxiliary encoder polarity. Users can set or clear this bit to change the Auxiliary encoder reading direction. When set to "0", do not invert aux encoder.

CG Bit (Zero Based)	Function	Description
	Feedback	When set to “1”, invert aux encoder.
9	Not in use	N/A
10	Analog Command Resolution Bit1	Currently Not used. Should be left 0.
11	Reserved	Should be left 0.
12 ÷ 15	Driver Type	12, 13, 14 – Set to "1" (Factory default).

Attributes:

Type:	Parameter.
Axis related:	Yes.
Array:	No.
Assignment:	Yes.
Command Allows Parameter:	---.
Scope:	All.
Restrictions:	Needs Motor OFF.
Save to Flash:	Yes.
Default Value:	0.
Range:	0 ÷ 65535.

Syntax:

```
XCG=0      ‘ Set X Axis CG=0.
YCG        ‘ Report value of CG for Y axis.
```

7.20 DA – Data Recording Array

Purpose:

"DA" is used to store data recording buffers. The "DA" array is automatically updated by the recorded data when the controller is in a data recording process.

When not used for data recording, the "DA" array can be used for any general purpose.

The DA and the AR arrays both share the same memory space. AR is limited to 1,000 from the communication only, but can be further used in order to access parameters with indexes larger than 1,000, via DA.

The actual size of DA is 16,000, but recordings are limited to 15,000. The data recordings are performed from the end of the DA vector, while ECAM[11], PixGen use the standard "AR" parameter, from the beginning.

Attributes:		
	Type:	Parameter.
	Axis related:	No.
	Array:	Yes, size = [1][16,000] –
	Recordings limited to 15,000.	
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	- 2,147,000,000 ÷ 2,147,000,000.

Syntax:

```
XDA[1]=0;          ' Set DA[1] "0".
XDA[1000]          ' Report value of DA[1,000].
YAR[300]=1000      ' Set DA[300]=1,000.
```

See also:

Data Recording (RG, RL, RV, RR, BR)

[11] ECAM Mode is not supported in the current FlexDC version.

7.21 DB – Download Buffer

The “DB” command provides an efficient fast download of large array buffers in CAN Bus Only.

Both the “DB” and “EDB” modes support auto-increment of the array index, meaning that the user only provides initial start index and then only sends the data.

7.22 DC – Deceleration

Purpose:

The normal Deceleration value from cruise velocity (towards Zero speed) in all motion modes that use the internal Profiler. This value is used to set the motion profile deceleration value in PTP, JOG, etc. The Deceleration value is defined in units of: [counts / sec^2] with resolution of 256 counts/sec^2.

The FlexDC support different deceleration values for normal deceleration and Limits (H/W or S/W) deceleration. See the “DL” parameter.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	100,000.
	Range:	512 ÷ 120,000,000.

Syntax:

```
XDC=1000000;    ‘ Set X Axis DC=1,000,000.
YDC=1000000;    ‘ Set Y Axis DC=1,000,000.
XDC             ‘ Report value of DC for X axis.
```

Examples:

The following code example shows starting a motion in X axis from Position “0” to Position “100,000”, using Speed Acceleration and Deceleration values.

```
XMO=1;XPS=0      ‘ Enables the motor and Set Position = “0”.
XMM=0;XSM=0      ‘ Set Normal Point To Point Motion Mode.
XAP=100000       ‘ Set Next PTP absolute location to “100,000” counts.
XAC=250000       ‘ Set Acceleration to “250,000”.
XDC=500000       ‘ Set Acceleration to “500,000”.
XSP=25000        ‘ Set Speed to “25,000”.
XBG              ‘ Start a Motion
```

7.23 DF – Download Firmware

Purpose:

To download any new firmware versions to the controller. New Firmware should be downloaded to the controller using the Shell utility application only.

WARNING! Under any circumstances, do NOT use the DF command directly from a terminal interface!

7.24 DL – Limit Deceleration

Purpose:

The Limit Deceleration value used by the profiler whenever one of the limits is detected (H/W or S/W) to stop from any speed to "0" speed. This value is used to set the motion profile Limit Deceleration value in PTP, JOG etc. The Limit Deceleration value is defined in units of: [counts / sec^2] with resolution of 256 counts/sec^2.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	100,000.
	Range:	512 ÷ 120,000,000.

Syntax:

```
ADL=2000000     ‘ Set DL=2,000,000 in all axes.
XDL             ‘ Report value of DL for X axis.
```

Examples:

The following code example shows starting a motion in X axis from Position "0" to Position "100,000". DL is set to 2,000,000 [counts / sec^2] (x 10 of AC and DC), so when the HL (High S/W Limit) is detected (at 50,000 counts), the servo controller stops the motion with deceleration of 2,000,000 [counts / sec^2].

```
XMO=1;XPS=0     ‘ Enables the motor and Set Position = “0”.
XMM=0;XSM=0     ‘ Set Normal Point To Point Motion Mode.
XAP=100000      ‘ Set Next PTP absolute location to “100,000” counts.
XAC=200000;XDC=200000   ‘ Set AC=DC= “200,000”.
XDL=2000000     ‘ Set Limit Deceleration to “2,00,000”.
XHL=50000       ‘ Set X High S/W Limit to “50,000” counts.
XSP=25000       ‘ Set Speed to “25,000”.
XBG             ‘ Start a Motion
```

7.25 DO – Analog DAC Offset

Purpose:

"DO" (Driver Command Offset) sets the Driver outputs command offset values. The Controller has two analog driver command outputs, one for X and one for Y.

The "DO" parameter sets the offset for the Driver command signals.

There is no special offset parameter for the analog outputs when used as general-purpose outputs (and not ad driver commands).

"DO" should be used only with the AB5 driver to calibrate the zero motion by using the "DO" command. "DO" is applied in LSB units. The range of the "DO" command is ±32,767 (±10 Volts). Normally the DO command is in the range of ±1600 (±0.5 Volts).

The value of "DO" is saved to the flash memory, and is restored on each power up. Note that "DO" has an effect whenever the system is powered on, regardless to the motor ON ("MO") and No Control ("NC") states. As a result, the offset calibration can be performed even when the controller is in Servo OFF state (MO=0).

"DO" is an axis-related parameter, and controls the offset of the various analog outputs as follows:

- "XDO" - Set the Analog Offset of the Main X Analog Command Channel.
- "YDO" - Set the Analog Offset of the Main Y Analog Command Channel.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	-32,767 ÷ 32,767.

Syntax:

```
XDO=100;      ‘ Set X DAC DO=100 (Offset = 30.5 mv).
XDO=-100;     ‘ Set X DAC DO=-100 (Offset = -30.5 mv).
XDO           ‘ Report value of AS for X axis.
BDO=0         ‘ Set DO=0 to both axes (no analog output offset).
```

See also: TC, AO.

7.26 DP – Desired Position

Purpose:

"DP" holds the actual instantaneous Desired Position or Reference Position Command of the servo control loop.

When an axis is not in motion, "DP" is constant and equals the local position reference point. When an axis is in motion, "DP" holds the real time servo loop control reference position. In standard Profiler based motions (e.g. Point to point, Jog, etc.), "DP" actually holds the Profiler position output value. Upon completing a standard Point to Point motion, "DP" holds the last value of "AP" used for that motion. In other motion modes, "DP" can be updated by other references (Analog input in Joystick mode, tables in ECAM mode, other axes in master slave modes, etc.).

When an axis servo loop is disables (MO=0), "DP" is continuously updated by the servo loop real time process to the value of "PS" (current encoder reading), so the Position Error ("PE") is "0" by definition.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	No.
	Command Allows parameter:	---.
	Scope:	---.
	Restrictions:	---.
	Save to Flash:	---.
	Default Value:	0.
	Range:	- 2,147,000,000 ÷ 2,147,000,000.

Syntax:

```
XDP     ‘ Report value of X axis DP.
ADP     ‘ Report value of DP to all axes.
```

See also:

AP, PS, PE.

7.27 EC – Communication Error Code

Purpose:

"EC" holds the last communication error code. The value of "EC" is reset to "0" when the controller boots up. When a communication error occurs (in one of the communication channels), the value of "EC" is updated accordingly by the Commands Interpreter to reflect the specific error cause.

The user can clear the value of EC to"0" at any time to clear the last Error Code register.

It should be noted that "EC" only holds errors generated by the Commands Interpreter if the source of the clause is communication. Errors generated by programs are reported by the "QC" parameter, and are "Program Task" specific.

Attributes:	**Type:**	Parameter.
	Axis related:	No.
	Array:	No.
	Assignment:	Yes (0 only).
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	0 ÷ 100.

Syntax:

```
XEC=0      ‘ Reset value of EC.
XEC        ‘ Report value EC.
```

7.28 EM – End of Motion Reason

Purpose:

"EM" is a read only parameter, reporting the last end of motion reason. "EM" is automatically updated by the real time controller firmware. The following End of Motion reasons are currently reported:

EM Value	EM Code	Description
0	EM_IN_MOTION	In motion, or After Boot up.
1	EM_NORMAL	Last Motion ended Normally.
2	EM_FLS	Last Motion ended due to Hardware FLS.
3	EM_RLS	Last Motion ended due to Hardware RLS.
4	EM_HL	Last Motion ended due to Software HL
5	EM_LL	Last Motion ended due to Software LL
6	EM_MF	Last Motion ended due to Motor Fault (check MF).
7	EM_USER_STOP	Last Motion ended due to User Stop (ST or AB).
8	EM_MOTOR_OFF	Last Motion ended due to Motor OFF (MO=0).
9	EM_BAD_PROFILE_PARAM	Last Motion ended due to Bad ECAM Parameters.

Table 22: End Of Motion Reason (EM) Codes.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	No.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	0 ÷ 9.

Syntax:

XEM ' Report value EM for X axis.

AEM ' Report value EM for all axes.

See also: MF.

7.29 ER – Max Position Error Limit

Purpose:

The “ER” parameter defines the Max allowed Positioning Error while the servo loop is enabled (MO=1).

The Positioning Error (“PE”) is defined as the current desired position minus the actual position: PE=DP-PS. The servo controller real time loop monitors the value of “PE” and compares it to the Max allowed error “ER”. When ABS(PE) > ER, the servo controller automatically disables the servo loop (switch automatically to MO=0 state).

The max allowed error “ER” is also monitored when the controller is in open loop mode (when NC=1 and MO=1), to avoid the motor from running over the end of travels. When the motor is disabled (MO=0) DP=PS, so the Position Error is “0” by definition.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	No.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	2000.
	Range:	1 ÷ 8,000,000.

Syntax:

```
XER=8000000;        ‘ Set X Axis ER to 8,000,000.
AER=2000            ‘ Set ER=2000 for all axes.
```

See also:

PS, DP, PE.

7.30 FF – Feed-Forward Gains

Purpose:

"FF" controls the Velocity and Acceleration Feed-Forward Gains.

The Velocity Feed-Forward gain is controlled by the FF parameter (FF[1]). The controller has an inherent Feed-Forward velocity, therefore normally the FF[1]=0. The Velocity Feed-Forward Gain (FF) is working on the profile velocity in counts/sec/8000 units. This feature is used only in a special mode of UHR with an AB1A Driver, and at a very low speed, less than 100µm/sec.

Command Acceleration Feed-Forward (Acc-FF) is controlled by the FF[2] parameter. The Acceleration Feed-Forward Gain (FF[2]) is working on the profile acceleration in counts/sec^2 / 2^{19} units.

In both cases, the resulted Feed-Forward value is added to the filter command output, in DAC [LSB] units.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	Yes, size = [2][2].
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	0 ÷ 65,536.

Syntax:

```
XFF[2]=200;        ' Set X Axis Acceleration FF to 200.
AFF=0              ' Set Velocity FF=0 for all axes.
```

Examples:

If motor's speed is 100 counts/sec (UHR Mode) and the user desires to add to the drive output the DC command of 0.5 V, then:

VFF=0.5*3276*8000/100=130400

XFF[1]=130400

7.31 HL – High Software Limit

Purpose:

"HL" is the Software High Position Limit. This value is monitored during all motions by the controller. Whenever the actual encoder position "PS" is higher then the "HL" value and the velocity "VL" is positive (moving towards higher positions), motion is stopped immediately using the stop deceleration parameter "DL".

"DL" should be normally set to a higher value then "DC", as during normal operation conditions "HL" is for emergency cases stop only.

The value of "HL" is validated by the controller during motion start "BG" commands only. i.e. a motion beyond the software limits (to an AP > HL) cannot be initiated, in motion mode Point-To-Point (MM=0). A special communication Error Code (EC=53) is generated by the BG command in that case (BG command returns ?>).

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	2,147,000,000.
	Range:	± 2,147,000,000.

Syntax:

```
XHL=100000000        ' Set X Software HL to 100,000,000
AHL=2147000000       ' Set Software HL to 2,147,000,000 for all axes.
```

See also:

DL, HL, PS, EC=53

7.32 IA – Indirect Array

Purpose:

"IA" is a user general-purpose Index Array. Although "IA" can be used for any general purpose during program development, it was intentionally defined to allow Indirect Index Addressing from within a script program.

The "IA" array is a non-axis related array, with a size of 100. Each element in the array is a LONG format number, which can be assigned to any value at any time. The index range of the "IA" array is: 1 ÷ 100. Since "IA" is non-axis related, accessing XAR, YAR, AAR, etc. actually access the same array element.

As noted, "IA" is a user general-purpose array, and is not used anywhere by the controller's firmware code, unless the user has included a reference to it within a script program.

Attributes:	**Type:**	Parameter.
	Axis related:	No.
	Array:	Yes, size = [1][53].
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	- 2,147,000,000 ÷ 2,147,000,000.

Syntax:

```
XIA[1]=0;        ‘ Set IA[1] “0”.
BIA[10]          ‘ Report value of IA[10] for X and Y
AIA[53]=1000     ‘ Set IA[53]=1,000.
```

Examples:

See below.

7.33 IL – Input Logic

Purpose:

The “IL” Input Logic parameter controls the logic of all digital inputs. Bits [0 : 23] of “IL” corresponds (and inverts) the relevant bits in “IP”. See the “IP” parameter reference for exact definitions of all “IP” and “IL” bits. By default IL=0. Each bit in “IL” that is assigned to “1” inverts the logic of the corresponding “IP” bit (bits [0 : 23] Only).

Notes:

- The ABORT input logic CANNOT be inverted. Being a SAFETY input, the ABORT logic must be configured such that when disconnected by the hardware, the ABORT is active, i.e. all axes are disabled.
- The Driver Fault Bits IP[24:25] can be inverted using CG[bit #6].

These are the “IL” parameter attributes:

Attributes:	**Type:**	Parameter.
	Axis related:	No.
	Array:	No.
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	0 ÷ 16,777,215 (0x00ff,ffff).

Syntax:

```
XIL          ‘ Report IL value (non-axis related).
AIL          ‘ Report IL value (non-axis related).
XIL=15       ‘ Inverts the logic of DIN1, DIN2, DIN3, DIN4.
```

See also:

CG, IP.

7.34 IP – Input Port

Purpose:

To read the digital Input Port bits of the FlexDC. The "IP" parameter is continuously updated by the real time servo loop to reflect the value of all digital input bits of the controller.

"IP" reports both the uncommitted digital inputs (Digital Inputs #1 ÷ #10), as well as all the committed digital inputs, i.e. limit switches, Driver Faults, and Abort input.

The FlexDC supports the following digital inputs (uncommitted and committed), according to the order as listed:

IP Bit# (0 Based) and Hex Value	H/W Signal Name / Functionality	IP Bit# (0 Based) and Hex Value	H/W Signal Name / Functionality
0 , 0x0000,0001	Digital Input #1 – Din1	16 , 0x0001,0000	X Axis – RLS
1 , 0x0000,0002	Digital Input #2 – Din2	17 , 0x0002,0000	X Axis – FLS
2 , 0x0000,0004	Digital Input #3 – Din3	18 , 0x0004,0000	Y Axis – RLS
3 , 0x0000,0008	Digital Input #4 – Din4	19 , 0x0008,0000	Y Axis – FLS
4 , 0x0000,0010	Digital Input #5 – Din5	20 , 0x0010,0000	X Index
5 , 0x0000,0020	Digital Input #6 – Din6	21 , 0x0020,0000	Y Index
6 , 0x0000,0040	Digital Input #7 – Din7	22 , 0x0040,0000	X Aux Encoder Index
7 , 0x0000,0080	Digital Input #8 – Din8	23 , 0x0080,0000	Y Aux Encoder Index
8 , 0x0000,0100	Digital Input #9 – Din9_Fast	24 , 0x0100,0000	X Axis – Driver Fault – Result – After Driver Fault Source logic (CG Mux)
9 , 0x0000,0200	Digital Input #10 – Din10_Fast	25 , 0x0200,0000	Y Axis – Driver Fault – Result – After Driver Fault Source logic (CG Mux)
10 , 0x0000,0400	X Axis External Fault Input	26 , 0x0400,0000	0
11 , 0x0000,0800	Y Axis External Fault Input	27 , 0x0800,0000	0
12 , 0x0000,1000	X Axis Internal Fault Input	28 , 0x1000,0000	ABORT Input
13 , 0x0000,2000	Y Axis Internal Fault Input	29 , 0x1000,0000	Not used.
14 , 0x0000,4000	X Index	30 , 0x2000,0000	Not used.
15 , 0x0000,8000	Y Index	31 , 0x4000,0000	Not used.

Notes:

- Bits #0 ÷ #9 of IP are the uncommitted Digital Inputs.

- Bits #16 ÷ #19 of IP are the RLS and FLS Limit Switch flags of axes X and Y
- RLS and FLS stand for the Reverse (Back) Limit Switch flag inputs, and the Forward (Front) Limit Switch flag inputs.
- The Driver Fault may be from 2 different sources:
- Internal Driver Fault Source – Used in Nanomotion Ltd Dedicated Drivers
- External Driver Fault Source – Used in 3rd party driver manufacturers.
- The Driver Fault source is set using CG[13].
- Bits #24 ÷ #25 of IP are the Driver Fault Inputs of axes X and Y. The polarity of these bits can be inverted using CG[6]. These bits are the result of the Driver Fault – Internal or external.
- The actual status of the internal or external Driver Fault may be found in Bits #12 ÷ #15.
- Bit #28 is the General Abort Input. When Abort is ON, all axes are disabled!
- The polarity of the ABORT bit **cannot be** inverted.
- Bits #29 ÷ #31 are currently not used.

These are the “IP” parameter attributes:

Attributes:	**Type:**	Parameter.
	Axis related:	No.
	Array:	No.
	Assignment:	No.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	0 ÷ 536,870,911 (0x1fff,fff).

Syntax:

XIP	‘ Report IP value (non-axis related).
YIP	‘ Report IP value (non-axis related).
AIP	‘ Report IP value (non-axis related).

See also:

IL, OP, Refer to "Part III– FlexDC Macro Language".

7.35 IS – Integral Saturation Limit

Purpose:

The “IS” parameter limits the output value of the Integral Term only when working in closed loop mode in PIV control scheme. “IS” limits ONLY the integral term saturation, and not the actual final control output, which is limited by the “TL” parameter. The purpose of “IS” is to allow different saturation limits to the Integral and control output. This is needed in some cases to avoid overshoots.

The range of “IS” is: 1 ÷ 32,767. IS=1 practically disables Integral term in the control filter. IS=32,767 is full range (100 % integral saturation).

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	32,763.
	Range:	1 ÷ 32,767.

Syntax:

```
XIS=16384;      ‘ Set X Axis IS=16,384 (50% of Max Range).
AIS=32767       ‘ Set IS=32,767 in all axes (100 % limit).
```

Examples:

The following code example shows starting a normal motion in X axis from Position “0” to Position “10,000”, but with the PID Integral term saturated to 25 %.

```
XMO=1;XPS=0          ‘ Enables the motor and Set Position = “0”.
XMM=0;XSM=0          ‘ Set Normal Point To Point Motion Mode.
XAP=10000            ‘ Set Next PTP absolute location to “100,000” counts.
XAC=90000;XDC=90000  ‘ Set AC=DC=90,000.
XSP=25000            ‘ Set Speed to “25,000”.
XIS=8192             ‘ Limit IS to 25 % (± 2.5 Volts)
XBG                  ‘ Start a Motion
```

See also: TL, Control Filter Implementation

7.36 KD – Control Filter Diff Term Gain

Purpose:

The “KD” parameter is used to set the control filter algorithm position loop Differential term gain in PID control mode, and Velocity loop overall gain in PIV control mode.

The “KD” parameter is an array parameter, with the size of [4]x[2], i.e. for each axis (X, Y), KD[1] and KD[2] are available. The first element “KD[1]” or “KD” (see note below) set the normal filter gains, while the second element “KD[2]” set the gain for the “Gain-Scheduling” algorithm.

Note: The FlexDC command interpreter supports (for backward compatibility) access to any array parameter first element, as a non-array element. This means that for example, “XKD” is identical to “XKD[1]”.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	Yes, size = [4][2].
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	32,767.
	Range:	0 ÷ 2,147,000,000.

Syntax:

```
XKD=16384       ‘ Set X Axis KD=16,384
XKD[1]=16384    ‘ Same as XKD=16384,  Set X Axis KD=16,384
XKD[2]=30000    ‘ Set X Axis KD[2]=30000 (for Gain Scheduling)
AKD=100000      ‘ Set KD=100,000 for all axes
```

See also:

CG, KP, KI

7.37 KI – Control Filter Integral Term Gain

Purpose:

The “KI” parameter is used to set the control filter algorithm position loop integral term gain in PID control mode, and the Velocity PI loop integral term gain in PIV control mode.

The “KI” parameter is an array parameter, with the size of [4]x[2], i.e. for each axis (X, Y), KI[1] and KI[2] are available. The first element “KI[1]” or “KI” (see note below) set the normal filter gains, while the second element “KI[2]” set the gain for the “Gain-Scheduling” algorithm.

Note: The FlexDC command interpreter supports (for backward compatibility) access to any array parameter first element, as a non-array element. This means that for example, “XKI” is identical to “XKI[1]”.

Attributes:		
	Type:	Parameter.
	Axis related:	Yes.
	Array:	Yes, size = [4][2].
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	32,767.
	Range:	0 ÷ 2,147,000,000.

Syntax:

```
XKI=16384       ‘ Set X Axis KI=16,384
XKI[1]=16384    ‘ Same as XKI=16384,  Set X Axis KI=16,384
XKI[2]=30000    ‘ Set X Axis KI[2]=30000 (for Gain Scheduling)
AKI=100000      ‘ Set KI=100,000 for all axes
```

See also:

CG, KP, KD

7.38 KP – Control Filter Proportional Term Gain

Purpose:

The “KP” parameter is used to set the control filter algorithm position loop proportional term gain in PID control mode, and the position loop overall gain in PIV control mode.

The “KP” parameter is an array parameter, with the size of [4]x[2], i.e. for each axis (X, Y), KP[1] and KP[2] are available. The first element “KP[1]” or “KP” (see note below) set the normal filter gains, while the second element “KP[2]” set the gain for the “Gain-Scheduling” algorithm.

Note: The FlexDC command interpreter supports (for backward compatibility) access to any array parameter first element, as a non-array element. This means that for example, “XKP” is identical to “XKP[1]”.

Attributes:		
	Type:	Parameter.
	Axis related:	Yes.
	Array:	Yes, size = [4][2].
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	32,767.
	Range:	0 ÷ 2,147,000,000.

Syntax:

```
XKP=16384      ‘ Set X Axis KP=16,384
XKP[1]=16384   ‘ Same as XKP=16384,  Set X Axis KP=16,384
XKP[2]=30000   ‘ Set X Axis KP[2]=30000 (for Gain Scheduling)
AKP=100000     ‘ Set KP=100,000 for all axes
```

See also:

CG, KI, KD

7.39 KR – Kill Repetitive Motions Command

Purpose:

The “KR” Kill Repetitive command terminates repetitive Point To Point motion cycles. Unlike the “ST” command, the motion is not stopped immediately, but stops after the current motion has ended.

Attributes:	**Type:**	Command.
	Axis related:	Yes.
	Array:	---.
	Assignment:	---.
	Command Allows Parameter:	No.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range :	---.

Syntax:

```
XKR;        ‘ Stop X Repetitive Motion
AKR         ‘ Stop Repetitive Motion of All axes.
```

Examples:

The Next example shows starting a Repetitive motion in X axis from Position “0” to Position “100,000” using “WT” Wait delay between the motions. KR is then issued to kill the repetitive motion.

```
XMO=1;XPS=0     ‘ Enables the motor and Set Position = “0”.
XMM=0;XSM=1     ‘ Set Repetitive Point To Point Motion Mode.
XAP=100000      ‘ Set Next PTP absolute location to “100,000” counts.
XAC=250000      ‘ Set Acceleration to “250,000”.
XDC=500000      ‘ Set Acceleration to “500,000”.
XSP=25000       ‘ Set Speed to “25,000”.
XWT=16384       ‘ Set 1 second delay between motions.
XBG             ‘ Starts a Motion
XKR             ‘ Terminatea the repetitive motion.
```

See also:

BG, AB, ST, SM, MM, WT

7.40 LD / SV – Load and Save Commands

Purpose:

"LD" and "SV" are the Load from Flash Memory and Save to Flash Memory commands. The "LD" and "SV" commands are used to load and save the controller parameters and script programs to and from the board Flash Memory.

The "LD" and "SV" commands can only be issued while all motors are in disable mode (in MO=0). SV should be issued only when the system is not in motion.

The "SV" command can receive the following parameters:

```
ASV      ' Save All Parameters and Script Program to Flash Memory
ASV,1    ' Save Only the Controller Parameters to the Flash Memory
ASV,2    ' Save Only the Script Program to the Flash Memory
```

The "LD" and "SV" commands have the following attributes:

Attributes:		
	Type:	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Command Allows Parameter:	Yes (see SV command above).
	Scope:	All.
	Restrictions:	All motors must be OFF.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

```
XSV      ' Save all Parameters and Script Program to Flash Memory
XSV,1    ' Save Only Parameters to Flash Memory
XSV,2    ' Save Only the Script Program to Flash Memory
BLD      ' Load Parameters and Script Program from Flash Memory
```

7.41 LL – Low Software Limit

Purpose:

"LL" is the Software Low Position Limit parameter. This value is monitored during all motions by the controller. Whenever the actual encoder position "PS" is smaller then the "LL" value and the velocity "VL" is negative (moving towards lower positions), motion is stopped immediately using the stop deceleration parameter "DL".

"DL" should be normally set to a higher value then "DC", as during normal operation conditions "LL" is for emergency cases stop only.

The value of "LL" is validated by the controller during motion start "BG" commands only. i.e. a motion beyond the software limits (to an AP < LL) cannot be initiated, in motion mode Point-To-Point (MM=0). A special communication Error Code (EC=53) is generated by the BG command in that case (BG command returns ?>).

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	2,147,000,000.
	Range:	± 2,147,000,000.

Syntax:

```
XLL=100000000     ' Set X Software LL to 100,000,000
ALL=2147000000    ' Set Software LL to 2,147,000,000 for all axes.
```

See also:

DL, HL, PS, EC=53

7.42 ME – Master Encoder

Purpose:

"ME" is the Master Encoder Definition for Gearing and ECAM motion modes.

Gearing and ECAM are motion modes where an axis follows another axis position with a pre-defined (fixed) ratio (in Gearing) or using user defined position tables (in ECAM). The "ME" parameter defines the master axis for that purpose.

Note: The master axis can be in motor ON or OFF (i.e. MO=1, or MO=0) states. In the later case, the Master's DP=PS, so using a disabled axis as a master axis, provides true encoder position tracking.

"ME" defines which axis is the Master axis for a given slave motion. The "ME" parameter can be any valid physical axis, as described below:

- The "ME" parameter can select between the following encoder inputs:
  ME=0 for X Axis Encoder.
  ME=1 for Y Axis Encoder.
  ME=2 for X Auxiliary Encoder Input.
  ME=3 for Y Auxiliary Encoder Input.

Attributes:		
	Type:	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	0 ÷3.

Syntax:

```
XMM=1;      ' Set X Master Encoder as Y
YMM         ' Report Master Encoder of Y Axis
AMM=0       ' Set All MM=0
```

See also:

FR, MM

7.43 MF – Motor Fault Reason

Purpose:

"MF" is a read only parameter reporting the last Motor Fault Reason. "MF" is automatically updated by the real time controller firmware.

As actual motor faults always causes an MO=0 condition (Motor Disable), the purpose of the "MF" parameter is to latch the cause of the last fault, from the time that the motor is disabled, usually the immediate fault cause disappears.

On the FlexDC, the Motor Fault Reason parameter holds encoded information about the actual fault cause as follows:

- Lower 16 bits of "MF" hold Motor Fault Reason as generated and set by the real time firmware.
- Upper 16 bits of "MF" holds the extended Motor Fault Reason, as latched by the hardware.

The lower 16 bits of "MF" represent a general fault cause number as defined in the following table:

Lower 16 Bits of MF:

MF Value	MF Code	Description
0	MF_NO_FAULT	None – Normal Operation.
1	MF_DRV_FLT	Fault caused buy a Driver Error for a specific axis (DRV_FLT H/W line was asserted).
2	MF_ABORT_INPUT	Fault caused buy the general Abort Input (ABORT H/W line was asserted).
3	MF_HIGH_ERR	Fault caused for a specific axis, when its Position Error "PE" is exceeding the allowed maximum Position Error for that axis (when : Abs(PE) > ER).
4	MF_MOTOR_STUCK	Fault caused for a specific axis when a Motor Stuck Condition is detected. Motor Stuck Condition is a condition when the servo command is saturated (reaching "TL") for more then 0.5 seconds, and no motion is detected.

Table 23: Motor Fault Cause Reasons - (MF) Codes in FlexDC.

The Upper 16 bits of "MF" represent extended fault source options, and they are set ONLY when the lower 16 bits of MF equals to "1" (i.e. the motor fault type is Driver Fault). The additional fault source information is defined in the following table:

Upper 16 Bits of MF (Valid Only When Low16Bits(MF)==1):

MF Bit# (0 Based)	Fault Source	MF Bit# (0 Based)	Fault Source
(16+) 0	Internal Type Driver Fault	(16+) 8	Not used (set to 0)
(16+) 1	External Type Driver Fault	(16+) 9	Not used (set to 0)
(16+) 2	Under Voltage Fault (PD-AT-2M)	(16+) 10	Not used (set to 0)
(16+) 3	Over Voltage Fault (PD-AT-2M)	(16+) 11	Not used (set to 0)
(16+) 4	Encoder A quad B Error	(16+) 12	Not used (set to 0)
(16+) 5	Not used (set to 0)	(16+) 13	Not used (set to 0)
(16+) 6	Not used (set to 0)	(16+) 14	Not used (set to 0)
(16+) 7	Encoder Disconnect Line Error	(16+) 15	Not used (set to 0)

Table 24: Extended Motor Fault Cause Reasons - (MF) Codes in FlexDC

Note: The extended (upper 16 bits) of "MF" codes are OR'ed with the "MF_DRV_FLT" when asserted.

These are the "MF" parameter attributes:

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	No.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	See above.

Syntax:

XMF ' Report Motor Fault for X axis.

AMF ' Report value of MF for all axes.

See also:

EM

7.44 MM – Motion Mode

Purpose:
To set the controller Motion mode for the next Motion. Currently, the following motion modes are supported:

MM=0 – Point to Point.
MM=1 – Jogging.
MM=2 – Position Based Gearing.
MM=5 – Position Based ECAM.
MM=8 – Step Command (no profiler).

The MM command is restricted to a No Motion condition. Trying to change the MM value while motion is in progress generates an “EC_NEEDS_MOTION_OFF” error # 50.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	Needs Motion OFF.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	0 ÷ 8.

Syntax:

```
XMM=1;      ‘ Set X Axis MM=1.
AMM=0       ‘ Set MM= 0 in all axes.
```

Examples:

The example shows starting a Normal (Non-Repetitive) motion in X axis from Position “0” to Position “100,000”.

```
XMO=1;XPS=0      ‘ Enables the motor and Set Position = “0”.
XMM=0;XSM=0      ‘ Set Normal Point To Point Motion Mode.
XAP=100000       ‘ Set Next PTP absolute location to “100,000” counts.
XAC=250000       ‘ Set Acceleration to “250,000”.
XDC=500000       ‘ Set Acceleration to “500,000”.
XSP=25000        ‘ Set Speed to “25,000”.
XBG              ‘ Start a Motion
```

The following code example shows starting a Jog motion in the Y axis using SP=-50,000 counts/sec (Negative Motion).

```
YMO=1;YPS=0      ‘ Enables the motor and Set Position = “0”.
YMM=1;YSM=0      ‘ Set Normal JOG Motion Mode.
YAC=250000       ‘ Set Acceleration to “250,000”.
YDC=500000       ‘ Set Acceleration to “500,000”.
YSP=-50000       ‘ Set Speed to “-50,000”.
YBG              ‘ Start a Motion
```

The next example shows a STEP motion in X axis from Position “0” to Position “100”. Note that in STEP motions there is no profile, so AC/SP/DC may not be set. When the BG command is issued, the reference position of the relevant axis is set immediately to the value of AP. Note that MM=8 can be combined with SM=1 to generate repetitive STEP motions.

```
XMO=1;XPS=0      ‘ Enables the motor and Set Position = “0”.
XMM=8;XSM=0      ‘ Set Normal STEP Motion Mode.
XBG              ‘ Start a Motion
```

See also:

SM, BG, WT, MO

7.45 MO – Motor ON (Enable / Disable the Servo Loop)

Purpose:

Set motor ON or OFF. MO=0 turns the relevant motor OFF (disabling the motor driver), and MO=1 sets the motor ON (enabling the driver). Note that when MO=1 command is issued, the DP (desired Position) is set to PS (actual position).

The FlexDC supports special hardware configurations for 1 or 2 axes mode. When, in a single axis configuration, an MO=1 assignment is given to the non-supported axis, a special Error Code is issued (EC=54, AXIS_NOT_SUPPORTED).

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	0 ÷ 1.

Syntax:

```
XMO=1;          ‘ Enable X motor (set XMO to 1).
AMO=0           ‘ Set MO=0 in all axes (Disable All Axes)
```

Examples:

The following code example shows starting a Jog motion in the Y axis using SP=-50,000 counts/sec (Negative Motion). MO must be set to “1” to start the motion.

```
YMO=1           ‘ Enables the motor and Set Position = “0”.
YPS=0           ‘ Set Position = “0”.
YMM=1;YSM=0     ‘ Set Normal JOG Motion Mode.
YAC=250000      ‘ Set Acceleration to “250,000”.
YDC=500000      ‘ Set Acceleration to “500,000”.
YSP=-50000      ‘ Set Speed to “-50,000”.
YBG             ‘ Start a Motion
```

See also: NC, BG, MM, SM, PS

7.46 MS – Motion Status

Purpose:

The “MS” Motion Status parameter holds information on the current motion status of specific axes. This is a read only, axis related parameter.

When an axis is not in motion, its “MS” parameter is “0” by definition, i.e. all bits are cleared (whether a motion did not start at all, or ended due to any reason).

The “MS” parameter is a bit field array. Each bit represents a certain motion status. More then one bit can be high i.e. logically “1” during a motion sequence.

“MS” is most commonly used to monitor the end of motion condition. Another way to monitor end of motion is to use the extended WAIT commands (“QW”). Refer to “Part III– FlexDC Macro Language” for more information.

The following table describes the current supported “MS” bits:

MS BIT	MS Code	MS Hex Value	Description
0	MS_IN_MOTION	0x00000001	Whenever this bit in “1”, the axis is in Motion.
1	MS_IN_STOP	0x00000002	This bit “1”, when the axis is stopping due to user command or any other non-normal stop condition, such as Limit, etc.
2	MS_IN_ACC	0x00000004	This bit indicates that the axis is accelerating.
3	MS_IN_DEC	0x00000008	This bit indicates that the axis is decelerating.
4	Reserved	0x00000010	Not used In This Version.
5	Reserved	0x00000020	Not used In This Version.
6	MS_IN_WAIT_REP	0x00000040	This bit indicates that the axis is waiting for the Wait Time to elapse in Repetitive PTP motions.
31-7	Reserved	0x00000080	Not used In this version.

Table 25: “MS” Motion Status Parameter Bits Description

The “MS” parameter has the following attributes:

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	No.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	0 ÷ N.A.

Syntax:

```
XMS          ‘ Report value MS for X axis.
AMS          ‘ Report value MS for all axes.
```

Examples:

The following simple example demonstrates a repetitive motion in the X axis, not using the internal Repetitive PTP mode, but rather by a simple script that polls the “MS” to check end of motion and to initiate a backward motion and so on.

```
XMO=1;XPS=0      ‘ Enables the motor and Set Position = “0”.
XMM=0;XSM=0      ‘ Set Normal Point To Point Motion Mode.
XAC=250000       ‘ Set Acceleration to “250,000”.
XDC=500000       ‘ Set Acceleration to “500,000”.
XSP=25000        ‘ Set Speed to “25,000”.

#X_START         ‘ Label for REP PTP Motion
            XAP=100000              ‘ Set Next PTP absolute location to “100,000”
            XBG ‘ Start a Motion
      @while (XMS != 0)    ‘ Wait for End Of Motion in X (XMS=0)
      @endwhile

            XAP=0        ‘ Set Next PTP absolute location to “0”
            XBG ‘ Start a Motion
      @while (XMS != 0)    ‘ Wait for End Of Motion in X (XMS=0)
      @endwhile

XJP, #X_START
```

See also: BG, EM, MM, SM, TR, TT.

7.47 NC – No Control (Set Open Loop Mode)

Purpose:

The “NC” parameter keyword set the controller to open-loop mode. In this mode the user can command a direct analog output command to the controller Analog Command ***(Acmd)*** output, bypassing the PIV controller filter. Refer to the “TC” command for further information.

The value of “NC” is not saved to the Flash Memory, and each time the controller boots up, the value of “NC” is set to “0” by default (normal closed loop operation). In order to switch to open loop mode, the user should switch the motor OFF (set MO=0 for the relevant axis), then set the value of “NC” to “1” (NC=1), and then switch the motor ON again. After MO=1 with NC=1, by default the analog output value commend is “0” to avoid motor motion (TC is set automatically to zero when MO=1). In this state, the user can control the actual analog output value by using the “TC” (Torque Command parameter keyword).

It should be noted that in open loop mode the actual analog command is still limited by the “TL” (Torque Limit) parameter. In addition, the control 2nd order filter may be used to monitor its operation and actual effect on the analog output value. The operation of the filter can be disabled by an appropriate flag (see 2nd order filter definitions in chapter 4, Part II). The user can choose to record the actual Driver Command (“PO”) value. In case the 2nd order filter is enabled, the actual value recorded is the step response of the filter. If no 2nd order filter is used, the actual value recorded is equal to the value commanded by “TC”. In any case the value is saturated by “TL”.

In order to switch back again to closed loop operation the motor should be disabled (MO=0), and only then “NC” may be set back to “0”.

The “NC” command is restricted to motor OFF condition. Trying to modify “NC” while motor is enabled (MO=1) generates an “EC_NEEDS_MOTOR_OFF” error # 48.

The “NC” parameter has the following attributes:

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	Needs Motor OFF.
	Save to Flash:	No.
	Default Value:	0.
	Range:	0 ÷ 3.

Syntax:

```
XNC=0;          ‘ Disable open loop (default)
ANC=1           ‘ Set NC=1 (open loop) for all axes.
```

Examples:

The following code example enables open loop mode on Y axis, and set the Y axis analog command output to +5 volts and –10 volts:

```
YMO=0           ‘ Must Disables the motor before changing the NC.
YNC=1           ‘ Set NC=1 to indicate open loop for that axis.
YMO=1           ‘ Set MO=1 for Y Again.
YTL=32763       ‘ Set Command saturation to ± 10 Volts.
YTC=16384       ‘ Set command value to +50% (+5 Volts).

YTC=-32763      ‘ Set command value to -100% (-10 Volts).

YMO=0           ‘ Disables the motor before changing the NC.
YNC=0           ‘ Restore closed loop mode.
```

See also:

TC, TL, 2nd order filter definitions, Data Recording

7.48 OC – Output Clear Bit Command

Purpose:

The “OC” command Clears (Set to “0”) a specific Bit in the digital Output Port word.

Unlike the “OP” parameter that only allows simultaneous access to all the output bits, the “OC” command allows bit wise clear operations on the digital output word.

This is required for example when only a certain bit needs to be cleared, without the other bits changed. Using the “OC” Output Clear Bit command saves the user from first reading the value of “OP”, clearing one of its bits using a logical “&” operator, and then re-assign “OP” (read-modify-write). When accessing the output port bits from two separate script tasks, this is necessary, otherwise the value of “OP” can be wrong.

The “OC” Output Clear Bit command must receive a parameter, indicating the specific bit to mask (currently: 1 ÷ 8). Calling the command without a parameter generates an “EC_PARAM_EXPECTED” (EC=38) error. Calling the command with an out of range parameter, generates an “EC_PARAM_OUT_OF_RANGE” (EC=34) error.

Attributes:		
	Type:	Command.
	Axis related:	No.
	Array:	No.
	Assignment:	---.
	Command Allows parameter:	Must have, Bit # (1 ÷ 8).
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Command Parameter Range:	1 ÷ 8.

Syntax:

```
XOC,1     ‘ Clears the first bit (LSB, Bit 0) in OP to “0”.
XOC,8     ‘ Clears the last bit (MSB, Bit 7) in OP to “0”.
```

Examples:

```
XOP=255   ‘ Set ALL digital outputs to High (“1”) logic.
XOC,1     ‘ Clears the first bit (LSB, Bit 0) in OP to “0” (OP=254)

XOP=255   ‘ Set ALL digital outputs to High (“1”) logic.
XOC,8     ‘ Clears the last bit (MSB, Bit 7) in OP to “0” (OP=127)
```

See also: OL, OP, OS.

7.49 OL – Output Logic

Purpose:

Sets (and gets) the FlexDC Output Port Logic control word. Using the "OL" Output Logic parameter, the user can control the actual H/W logic level of each bit in the controller Output Port Word.

Each bit in "OL" corresponds to the same bit in "OP", and to a specific H/W digital output. The bit order of "OL" is the same as "OP", i.e.:

- Bit 0 of OL – Controls the logic of digital output port #1.
- Bit 1 of OL – Controls the logic of digital output port #2.
- …
- Bit 7 of OP – Controls the logic of digital output port #8.

"OL" is non-axis related, so axis-preceding character has no effect. "OL" is usually set to a pre-defined value after the initial application setup, and then "OP", "OS" or "OC" should be used to control the outputs.

Attributes:	**Type:**	Parameter.
	Axis related:	No.
	Array:	No.
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	0 ÷ 255.

Syntax:

```
XOL=0;      ‘ Set non-inverted logic to all digital Outputs.
AOL         ‘ Report value of OL, the output port word.
XOL=128     ‘ Invert the logic of output port #8.
XOL=255     ‘ Invert the logic of all output ports.
```

See also: OC, OP, OS

7.50 OM – I/O Modes Hardware Configuration

Although “OM” is an axes related parameter (and it is implemented as such), in the FlexDC firmware there is no actual relation between the XOM, YOM, etc. to actual axes. The distinct axes identifiers are used in this case only to access more then one optional hardware registers of the FlexDC. Writing to “OM” immediately changes the corresponding internal hardware register values.

In the current firmware version, there are only 2 functional registers related to the “OM” parameter. These are in Table 26:

“OM” Axis #	Hardware Register	Functionality
XOM	IO_MODE_0	Controls Digital Outputs Assignment (as normal or Compare Output functions).
YOM	IO_MODE_1	Controls Encoder Capture I/O signal Source and logic.

Table 26: "OM" - I/O Mode Configuration Functionality Definitions

IO_MODE_0 – “XOM”:

Table 27 describes the **IO_MODE_0** bits order and Table 28 describes the specific description:

IO_MODE_0 : Bit # 31 ÷ 0																													
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved																												**Out 6 Src**		**Out 5 Src**	

Table 27: IO_MODE_0 Bits Order

Bits	Name	IO_MODE_0 Bits Description
1 .. 0	**DOut5 Src**	Defines the Dout5_Fast (fast output) source: 00 – As normal Output. 01 – From compare X 10 – From compare Y
3 .. 2	**DOut6 Src**	Defines the Dout6_Fast (fast output) source: 00 – As normal Output. 01 – From compare X 10 – From compare Y
31 .. 4	**Reserved**	These bits are currently not used, and should be left “0” for future compatibility.

Table 28: FlexDC "XOM" - IO_MODE_0 Bits Configuration Description

IO_MODE_1 – “YOM”:

Table 29 describes the **IO_MODE_1** bits order and Table 30 describes specific description:

IO_MODE_0 : Bit # 31 ÷ 0																													
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved																			**YPOL**	**Y Axis - Capture Event Source Def**			**Reserved**			**XPOL**	**X Axis - Capture Event Source Def**		

Table 29: IO_MODE_1 Bits Order

Bits	Name	IO_MODE_0 Bits Description
7 .. 0	**X Axis Evt**	Bits **[3–0]** of **IO_MODE_1** controls the Capture Event Source for the **X Axis**. The following bit order definitions applies: “0000” X Event source is Din1. “0001” X Event source is Din2. “0010” X Event source is Din3. “0011” X Event source is Din4. “0100” X Event source is Din5. “0101” X Event source is Din6. “0110” X Event source is Din7. “0111” X Event source is Din8. “1000” X Event source is Din9_Fast. “1001” X Event source is Din10_Fast. “1010” X Event source is Index X. “1011” X Event source is Index Y.
	X Pol	Select Input polarity for Axis X. “0” set Normal pulse polarity, “1” set Inverted pulse polarity.
7 .. 5	**Reserved**	These bits are currently not used, and should be left “0” for future compatibility
11 .. 8	**Y Axis Evt**	Bits **[11–8]** of **IO_MODE_1** controls the Capture Event Source for the **Y Axis**. The following bit order definitions applies: “0000” Y Event source is Din1. “0001” Y Event source is Din2. “0010” Y Event source is Din3. “0011” Y Event source is Din4. “0100” Y Event source is Din5. “0101” Y Event source is Din6. “0110” Y Event source is Din7. “0111” Y Event source is Din8. “1000” Y Event source is Din9_Fast. “1001” Y Event source is Din10_Fast. “1010” Y Event source is Index Y. “1011” Y Event source is Index X.
12	**Y Pol**	Select Input polarity for Axis Y. “0” set Normal pulse polarity, “1” set Inverted pulse polarity.
31..13	**Reserved**	These bits are currently not used, and should be left “0” for future compatibility

Table 30: FlexDC "YOM" - IO_MODE_1 Bits Configuration Description

The “OM” parameter has the following attributes:

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	-2,124,000,000 ÷ +2,124,000,000.

Syntax:

```
XOM=0        ‘ Reset IO_MODE_0 register.
XOM          ‘ Report Value of IO_MODE_0.
XOM=0        ‘ Reset IO_MODE_1 register.
XOM          ‘ Report Value of IO_MODE_1.
```

Examples:

The following assignment set all digital outputs as standard normal outputs, controlled by the “OP” parameter.

```
XOM=0
```

The following assignment set Digital Output #5 (DOut5) to be assigned to X Axis Compare, Digital Output #6 (DOut6) to be assigned to Y Axis Compare. In this MODE accessing the bits 5 and 6 of “OP” (by modifying “OP” value or with the “OC” and “OS” Output Clear and Set Bit commands) only modifies the value of “OP”, but does not affect the actual hardware output pins.

```
XOM=9
```

The following assignment set only Digital Output #5 (DOut5) to be assigned to X Axis Compare. All other digital outputs as standard normal outputs, controlled by the “OP” parameter.

```
XOM=1
```

The following assignment set only Digital Output #5 (DOut1) to be assigned to Y Axis Compare. All other digital outputs as standard normal outputs, controlled by the “OP” parameter.

```
XOM=2
```

The following assignment defines the X Axis Capture Source to be the X Encoder Index Input.

YOM=10

The following example demonstrates simultaneous independent usage of X and Y axes, axes Compare and Capture functions.

- The X axis is configured to generate Compare pulses on DOut5, and assuming that DOut5 is connected by external wiring to Din9, the X Capture function is programmed to latch the Compare locations.
- The Y axis is configured to generate Compare pulses on DOut6, and assuming that DOut6 is connected by external wiring to Din10, the Y Capture function is programmed to latch the Compare locations.

```
                ' Set X Compare to DOut5  and Y Compare to DOut6.
                ' The resulted value is: 5.
                ' ------------------------------------------------------------------------
        '
XOM=9           ' Set IO_MODE_0
YOM=2312        ' Set IO_MODE_1:X Capture on Din9, Y Capture on Din10.
```

See also:

OP, IP, Compare Function, Capture Function, refer to the "Technical Data" chapter in the "FlexDC User Manual" regarding the Fast Digital Outputs and Inputs.

7.51 OP – Output Port

Purpose:

Sets (and gets) the FlexDC Motion Controller's uncommitted digital Output Port bits.

The FlexDC servo controller supports 6 general-purpose digital outputs (refer to the "Technical Data" chapter in the "FlexDC User Manual" for more information about hardware interfaces of digital I/O).

The "OP" parameter holds the Output Port word (bit array). Each bit in "OP" controls a single digital output bit port (as shown below). The user can of course read the value of "OP" in order to get the current Output Port word status.

- Bit 0 of OP – Controls digital output port #1.
- Bit 1 of OP – Controls digital output port #2.
- Bit 7 of OP – Controls digital output port #8.

"OP" controls simultaneous access to all the Output Port word bits at one assignment. In order to access one bit at a time (Set or Clear a specific bit), the FlexDC firmware includes 2 additional commands: "OS" – That Set (to "1" logic) a specific output bit, and "OC" – That clears (to "0" logic) a specific output bit. See "OS" and "OC" references. The user can also control the actual H/W logic level of each output bit using the "OL" – Output Logic parameter. "OP" is non-axis related, so axis-preceding character has no effect.

Attributes:	**Type:**	Parameter.
	Axis related:	No.
	Array:	No.
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	0 ÷ 255.

Syntax:

```
XOP=0;      ‘ Set the Output Port to “0” (all bits cleared).
AOP         ‘ Report value of OP, the output port word.
XOP=255     ‘ Set ALL digital outputs to High (“1”) logic.
```

See also:

OC, OL, OS, XOM.

7.52 OS – Output Set Bit Command

Purpose:

The “OS” command sets (Set to “1”) a specific Bit in the digital Output Port word.

Unlike the “OP” parameter that only allows simultaneous access to all the output bits, the “OS” command allows bit wise set operations on the digital output word.

This is required for example when only a certain bit is need to be set, without changing the other bits. Using the “OS” Output Set Bit command saves the user from first reading the value of “OP”, setting one of its bits using a logical “|” operator, and then re-assign “OP” (read-modify-write). When accessing the output port bits from two separate script tasks, this is necessary, otherwise the value of “OP” can be wrong.

The “OS” Output Set Bit command must receive a parameter, indicating the specific bit to set (currently: 1 ÷ 8). Calling the command without a parameter generates an “EC_PARAM_EXPECTED” (EC=38) error. Calling the command with an out of range parameter, generates an “EC_PARAM_OUT_OF_RANGE” (EC=34) error.

Attributes:		
	Type:	Command.
	Axis related:	No.
	Array:	No.
	Assignment:	---.
	Command Allows parameter:	Must have, Bit # (1 ÷ 8).
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Command Parameter Range:	1 ÷ 8.

Syntax:

```
XOS,1      ‘ Sets the first bit (LSB, Bit 0) in OP to “1”.
XOS,8      ‘ Sets the last bit (MSB, Bit 7) in OP to “1”.
```

Examples:

```
XOP=0      ‘ Clears ALL digital outputs to Low (“0”).
XOS,1      ‘ Set the first bit (LSB, Bit 0) in OP to “1” (OP=1)

XOP=0      ‘ Clears ALL digital outputs to Low (“0”).
XOS,8      ‘ Sets the last bit (MSB, Bit 7) in OP to “1” (OP=128)
```

See also:

OC, OL, OP

7.53 PA – Parameters Array

Purpose:

"PA" is a user general-purpose parameters array. "PA" can be used during script program development for any purpose.

The "PA" array is an axis related array, with a size of 2x100 elements. Each element in the array is a LONG format number, which can be assigned with any value at any time. The index range of the "PA" array is: 1 ÷ 100.

As noted "PA" is a user general-purpose array, and is not used anywhere by the controller's firmware code, unless the user has included a reference to it within a script program.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	Yes, size = [2][200].
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	- 2,147,000,000 ÷ 2,147,000,000.

Syntax:

```
XPA[1]=0;          ‘ Set XPA[1] “0”.
YPA[10]            ‘ Report value of YPA[10]
BPA[100]=1000      ‘ Set both axes : PA[100]=1,000.
```

Examples:

See below.

7.54 PE – Position Error

Purpose:

"PE" is a read only parameter, holding the actual servo loop positioning error.

The Positioning Error ("PE") is defined as the current desired position minus the actual position:

PE=DP-PS.

Whenever the servo loop is enabled (MO=1) in both open and closed loop modes, the real time software computes and updates the value "PE". When the motor is disabled (MO=0) DP=PS, so the Position Error is "0" by definition.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	No.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	- 8,000,000 ÷ 8,000,000.

Syntax:

XPE	' Report X axis Positioning Error PE.
APE	' Report Positioning Error PE for all axes.

See also:

ER, PS, DP.

7.55 PG – Position Compare Parameters Array

Purpose:

The “PG” array elements control the operation of the Position Compare Function. “PG” is an axis related array, sized [2 x 8]. Each axis has 8 parameters controlling the compare function operation as described below. The FlexDC product supports the Compare Function on both axes X and Y.

The “PG” array parameter has the following attributes:

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	Yes, size = [2][8].
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	- 2,147,000,000 ÷ 2,147,000,000.

See Compare Function description for full limitations description.

Syntax:

```
XPG[1]=0      ‘ Set X Axis PG[1] to “0” (set X axis Mode 0).
YPG[2]=100    ‘ Set Y Axis PG[2] to “100” (set Y axis Compare Distance=100).
XPG[7]=0      ‘ Set X Axis PG[7] to “0” (set W axis Compare Pulse Polarity).
```

See also:

PQ

7.56 PQ – Compare Function Activate / Disable Command

Purpose:

The “PQ” command is an axis-related command, enabling or disabling the Compare Function for a specific axis. The command requires a parameter indicating the requested operation. The command syntax is as follows:

XPQ,*Parameter*

where:

- X is an axis identifier.
  For the current **FlexDC** version the compare function is supported both axes X and Y.
- ***Parameter=0:*** Indicates immediate disable of compare for the specified axis. No conditions are checked expect a valid axis identifier.
- ***Parameter=1:*** Indicates start compare function for the specified axis. The command validates correct parameter (“PG”) for the specific requested mode.

In any case that one of the command’s parameters is out of range, the command returns an error prompt: “?>” or generates a script “Run Time Error” (if called from within a script macro program). The relevant Error Code flags (“EC” or “QC”) is updated to reflect the error cause.

Notes:

- The user should be aware that not all conditions for the correct operation of the Compare Function can be validated during command initialization. For example, the minimal distance between each two consecutive points in the “AR” table (in Modes 2 and 3) cannot be tested as the limitation depends on the actual motion speed.
- It is the user’s responsibility to specify correct parameters values for each of the supported Compare Modes.

The “PQ” command has the following attributes:

Attributes:	**Type:**	Command.
	Axis related:	Yes.
	Array:	---.
	Assignment:	---.
	Command Allows Parameter:	Must Have, Number (0, or 1).
	Scope:	All.
	Restrictions:	See above.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

The command syntax is as follows (see also syntax definitions above):

```
XPQ,1          ‘ Enable Compare Function for X Axis.
YPQ,1          ‘ Enable Compare Function for Y Axis.
```

See also:

PG

7.57 PO – PIV Output

Purpose:

"PO" is a read only parameter reflecting the actual servo driver command value.

In closed loop operation, "PO" is the actual servo control output. In open loop operation, "PO" equals the "TC" command.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	No.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	---.
	Save to Flash:	---.
	Default Value:	---.
	Range:	- 32,767÷ 32,767.

Syntax:

```
XPO     ‘ Report PO value for X Axis
APO     ‘ Report PO value for all Axes
```

See also:

TC, and Control Loop Description

7.58 PS – Position (Encoder Position)

Purpose:

This command reports the actual controller position (Encoder Value). The user can also be set as desired value to the current position (define the current position as ##). Note that setting the position value is valid only when not in motion. Setting the "PS" immediately sets the "DP" (desired position) to the same value.

The "PS" command is restricted to No Motion condition. Trying to change "PS" value while motion is in progress generates an "EC_NEEDS_MOTION_OFF" error # 50.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	Needs Motion OFF.
	Save to Flash:	No.
	Default Value:	0.
	Range:	- 2,147,000,000 ÷ 2,147,000,000.

Syntax:

```
XPS=0;          ' Set X Axis Position (encoder) to "0"
APS=0           ' Set PS=0 in all axes (Reset All Axes)
```

Examples:

The following example shows resetting the X axis position to "0', and then initiate a normal motion in X axis from Position "0" to Position "100,000".

```
XMO=1           ' Enables the X Motor
XPS=0           ' Set X axis encoder Position = "0".
XMM=0;XSM=0     ' Set Normal Point To Point Motion Mode.
XAP=100000      ' Set Next PTP absolute location to "100,000" counts.
XAC=250000      ' Set Acceleration to "250,000".
XDC=500000      ' Set Acceleration to "500,000".
XSP=25000       ' Set Speed to "25,000".
XBG             ' Start a Motion
```

See also:

DP, MM, ER

7.59 RA – CAN Receiving Address

Purpose:

To set the CAN Receiving Address. The CAN Receiving address is the CAN address which the controller monitors for incoming CAN messages. Responses are sent to the CAN address defined by the "TA" parameter.

The CAN Receiving Address must be saved to the flash memory, and the controller must be reseated in order to change the CAN settings.

Changing RA/TA immediately re-initializes the CAN hardware to take the requested effect. Care should be taken, as changing RA/TA while working in CAN bus stops the communication with the PC. The parameters must still be saved to the Flash Memory (as in previous revisions) in order to be valid after boot.

A new Error Code "EC_HW_INIT_ERROR=97" was added to indicate a CAN hardware initialization error.

The **FlexDC**, in addition and independent to the standard RA and TA CAN addresses, listens and transmits on additional addresses. See the "ZI" keyword for more information

Attributes:		
	Type:	Parameter.
	Axis related:	No.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	1.
	Range:	0 ÷ 2047.

Syntax:

```
XRA=1        ' Set CAN  RA=1.
XRA          ' Report value of RA.
```

See also:

CB, TA, ZI

7.60 RG – Data Recording GAP

Purpose:

The “RG” Recording Gap parameter controls the number of servo cycles interval (Gap) between each two consecutive recorded data points.

The FlexDC data recording capabilities allows for collecting data at the servo loop rate, i.e. 8,192 per second. However, since currently the recording buffers are limited to 1,875 data points to each vector (up to 8 vectors simultaneously), at 8,192 points per second this would have limited the recording time to less then 1 second. In order to allow longer recordings “RG” is defined. For example, if RG=8, i.e. a data point is collected to the recording buffer each 8 servo cycles (i.e. at a rate of ~ 1msec per point), recording of up to 10 seconds is possible, and so on.

Attributes:	**Type:**	Parameter.
	Axis related:	No.
	Array:	Yes, Size [2].
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	1.
	Range:	1 ÷ 16,384.

Syntax:

```
XRG=1        ‘ Set Recording Gap to “1”.
XRG          ‘ Report value of RG.
```

See also:

BR, DA, RL, RR, RV

7.61 RG[2] – Data Recording Upload Delays

Purpose:
The “RG[2]” Recording Upload Delay parameter controls the number of servo cycles delay between each two consecutive CAN messages during Upload Recording Data in CAN bus operation mode.

When uploading large data buffers in CAN bus, the FlexDC can generate high loads on the CAN bus network. Depending on the PC load and type of CAN board, on high buffers upload, some CAN messages can be lost. In order to avoid this problem, the FlexDC can add delays between CAN messages during data recording upload. The Delay is set by RG[2], and is given in servo sample time multipliers.

RG[2]=0 means no delay. RG[2]=1 means 1 sample time delay (this is 61 micro-sec on the 4M and 122 micro-sec on the 2M) and so on.

Usually, a delay of 3-5 samples is sufficient for most cases.

For complete description of the RG keyword attributes and examples see the RG keyword command reference above.

Attributes: See “RG” keyword above.

Syntax: See “RG” keyword above.

Examples: See “RG” keyword above.

7.62 RL – Data Recording Length

Purpose:

The “RL” Recording Length parameter controls the number of data points to be collected to the recording buffers during data recording process, and as a result the overall recording time.

The “RL” parameter defines the number of points per vector. If RL=1000, this means that for each selected vector to be recorded, 1,000 data points are collected. The total number of points collected in the recording process is: RL x Number of Recorded Variables. Currently, the FlexDC supports up to 8 recorded vectors of up to 1,875 points each, to a total of 15,000 data points overall.

The overall data recording time is: (RL x RG) / 8,192 in [sec] units.

Attributes:	**Type:**	Parameter.
	Axis related:	No.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	1.
	Range:	1 ÷ 15,000.

Syntax:

```
XRL=1000        ‘ Set Recording Length to “1,000”.
XRL             ‘ Report value of RL.
```

See also:

BR, DA, RG, RR, RV

7.63 RP – Relative Position

Purpose:

The RP parameter defines the next motion Relative Position (in counts) target.

The relative position is used for Relative Point to point motions. When issuing an RP=## command the value of the next absolute position is computed as follows: AP=DP+RP. Upon a BG (begin motion) command, the controller generates a profile from the current desired ("DP") position to the current "AP". Refer to the "AP" command for more information.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	- 2,147,000,000 ÷ 2,147,000,000.

Syntax:

```
XRP=100000;      ' Set X Axis Relative Position to "100,000".
ARP=100          ' Set RP=100 in all axes.
```

Examples:

The following example shows performing a +100 counts step, followed by a –100 counts step:

```
XMO=1            ' Enables the X motor
XPS=0            ' Set X axis encoder Position = "0".
XMM=0;XSM=0      ' Set Normal Point To Point Motion Mode.
XAC=90000;XDC=90000   ' Set AC=DC=90,000.
XSP=25000        ' Set Speed to "25,000".
XRP=100          ' Define a +100 counts step.
XBG              ' Start a Motion

XRP=-100         ' Define a -100 counts step.
XBG              ' Start a Motion
```

See also: DP, AP, PS, BG

7.64 RR – Data Recording Status

Purpose:

"RR" is a read only parameter, indicating the recording process status.

When a new recording begins (after "BR" command is issued) "RR" is internally set to the value of "RL". During the data recording process, "RR" is automatically decremented by "1" for each data point collected (to all buffers). This practically happens every "RG" servo cycles. When "RR" equals "0", data recording has terminated, and the recorded data can be uploaded.

When RR > 0, data recording upload is denied.

Attributes:	**Type:**	Parameter.
	Axis related:	No.
	Array:	No.
	Assignment:	No.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	---.
	Range:	1 ÷ 10,000.

Syntax:

XRR ' Report value of RR.

See also:

BR, DA, RG, RL, RV

7.65 RS – Reset Controller Command

Purpose:

The “RS” command can be used to reset the controller software.

“RS” causes the FlexDC micro-processor to enter a software reset state, and completely re-initializes the controller software.

After Reset, all the controller parameters and script program resume their boot up values. The AUTOEX starts running like in power on condition.

The “RS” command has the following attributes:

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Command Allows Parameter:	No.
	Scope:	All.
	Restrictions:	All motors must be OFF, no program running.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

```
XRS     ‘ Resets the FlexDC.
```

7.66 RV – Data Recording, Recorded Variables

Purpose:

The “PA” array is an axis related array, with a size of 2x100 elements. Each element in the array is a LONG format number, which can be assigned with any value at any time. The index range of the “PA” array is: 1 ÷ 100.

The “RV” keyword – Recorded variables, in the FlexDC is a non-axis related array, with the size of 1x8 elements. Using the “RV” array, the user may select the data member to be recorded for each one of the 8 data recording vectors. XRV[1] controls Vector #1, XRV[2] controls Vector #2, and so on.

The user can select one of 51 internal data members for each vector. In general the user can select one of 20 axis specific (currently 11 available and 9 reserved) data elements for each axis, and 10 global registers. In the following list all options for “RV” are defined. In the table below (i) indicates the requested axis in zero based form. X axis is defined by i=0, Y axis is defined by i=1.

RV Value	Data Member to be recorded for Axis (i)	Keyword
0	None (empty)	---
I x 20 + 1	Position	(i) PS
I x 20 + 2	Velocity	(i) VL
I x 20 + 3	Position Error	(i) PE
I x 20 + 4	Desired Position	(i) DP
I x 20 + 5	PID Output	(i) PO
I x 20 + 6	Status Register	(i) SR
I x 20 + 7	Motion Status	(i) MS
I x 20 + 8	Analog Input	(i) AI
I x 20 + 9	Motor Fault	(i) MF
I x 20 + 10	Auxiliary Position	(i) XP
I x 20 + 11	Auxiliary Velocity	(i) XV
I x 20 + (12 ÷ 20)	Axis Related Reserved	---
41	Input Port	IP
42	Output Port	OP
43 ÷51	Reserved	---

The “RV” command has the following attributes:

Attributes:	**Type:**	Parameter.
	Axis related:	No.
	Array:	Yes, size = [1][8].
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	0 ÷ 211.

Syntax:

```
XRV[1]=0        ‘ Set X axis RV to 0 (no recording).
XRV[1]          ‘ Report value of RV[1].
```

See also:

BR, DA, RG, RL, RR

7.67 SM – Special Motion Mode Attribute Parameter

Purpose:

Defines an enhancement to the standard Point to Point Motion Mode (MM=0).

The following Special Modes are supported:

SM=0: No Special Mode**.**

SM=1, Repetitive Motion: Repetitive Point to Point. When the controller is in MM=0 (PTP) and SM=1, the motion is repetitive. This means that the axis is commanded to perform a PTP motion to the specified absolute position and then, after the motion is completed and a user specified delay (WT) is expired, a new motion is automatically initiated to the starting position (AP is updated to this value). When the latter motion is completed, and the WT delay is finished, the cycle starts again. This back-and-forth motion is repeated until stopped by one of the following clauses: AB (abort), ST (stop), KR (Kill repetitive), and MO=0.

The SM command is restricted to No Motion condition. Trying to change SM value while motion is in progress generates an “EC_NEEDS_MOTION_OFF” error # 50.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	Needs Motion OFF.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	0 ÷ 8.

Syntax:

```
XSM=1;        ‘ Set X Axis SM=1.
ASM=0         ‘ Set SM=0 in all axes.
```

Examples:

The following code example shows starting a Normal (Non-Repetitive) motion in X axis from Position “0” to Position “100,000”.

```
XMO=1;XPS=0     ‘ Enables the motor and Set Position = “0”.
XMM=0;XSM=0     ‘ Set Normal Point To Point Motion Mode.
XAP=100000      ‘ Set Next PTP absolute location to “100,000” counts.
XAC=250000      ‘ Set Acceleration to “250,000”.
XDC=500000      ‘ Set Acceleration to “500,000”.
XSP=25000       ‘ Set Speed to “25,000”.
XBG             ‘ Start a Motion
```

The Next example shows starting a Repetitive motion in X axis from Position “0” to Position “100,000” (same motion parameters as above), using “WT” Wait delay between the motions.

```
XMO=1;XPS=0     ‘ Enables the motor and Set Position = “0”.
XMM=0;XSM=1     ‘ Set Repetitive Point To Point Motion Mode.
XAP=100000      ‘ Set Next PTP absolute location to “100,000” counts.
XAC=250000      ‘ Set Acceleration to “250,000”.
XDC=500000      ‘ Set Acceleration to “500,000”.
XSP=25000       ‘ Set Speed to “25,000”.
XWT=16384       ‘ Set 1 second delay between motions.
XBG             ‘ Start a Motion
```

See also:

MM, WT, AB, ST, KR, MO

7.68 SP – Speed

Purpose:
Sets the Speed of the profile in PTP motions, and the Jogging speed in Jogging motions. The Speed value is defined in units of: [counts / sec]. The value of SP can be negative, to define a negative JOG motion. However, in PTP motion mode, the SP sign is ignored, and actual speed direction is set by position profile requirements.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	100,000.
	Range:	-30,000,000 ÷ 30,000,000.

Syntax:

```
XSP=100000;     ‘ Set X Axis SP=100,000.
ASP=20000       ‘ Set SP=200,000 in all axes.
```

Examples:
The following code example shows starting a Jog motion in the Y axis using SP=-50,000 counts/sec (Negative Motion).

```
YMO=1;YPS=0     ‘ Enables the motor and Set Position = “0”.
YMM=1;YSM=0     ‘ Set Normal JOG Motion Mode.
YAC=250000      ‘ Set Acceleration to “250,000”.
YDC=500000      ‘ Set Acceleration to “500,000”.
YSP=-50000      ‘ Set Speed to “-50,000”.
YBG             ‘ Start a Motion
```

See also: C, DL, SP, MM, BG

7.69 ST – Stop Motion Command

Purpose:

The “ST” Stop command stops any motion using the “DC” (deceleration value). Unlike the Abort (“AB”) command, the stop command stops the motion by generating a deceleration profile to “0” speed until a complete motion stop.

“ST” may be used whenever a motion needs to be stopped in controlled manner. For example, when a motion to search some input flag is performed, when the input is detected, the “ST” command may be used to stop the motion (see example below).

Attributes:	**Type:**	Command.
	Axis related:	Yes.
	Array:	---.
	Assignment:	---.
	Command Allows Parameter:	No.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

```
XST;        ‘ Stop X Motion
AST         ‘ Stop motion of All axes.
```

Examples:

The following example starts a motion, and then enters a loop to check for Input #1 to become low (“0”). When condition is met, the motion is stooped.

The following example can be written as a script program file. The main routine name is “#FINDI1”, and can be executed and tested.

```
‘
‘ Routine to find Input #1
‘ --------------------------
#FINDI1
‘
XMO=1;XPS=0              ‘ Enables the motor and Set Position = “0”.
XMM=0;XSM=0              ‘ Set Normal Point To Point Motion Mode.
XAP=100000               ‘ Set Relative motion of “100,000” counts.
XAC=90000;XDC=90000      ‘ Set AC=DC=90,000
XSP=25000                ‘ Set Speed to “25,000”.
XBG                      ‘ Start a Motion
‘
‘ Now enter a loop to check for input #1 to become low.
‘ ------------------------------------------------------------
@while (AIP & 1)         ‘ Wait for Input “1” to be Low.
@endwhile
‘
‘ Input is found, so stop the motion.
‘ ---------------------------------------
XST
‘
‘ Wait for End Of Motion
‘ ---------------------------
@while (XMS != 0)          ‘ Wait for MS (Motion Status) top be “0”.
@endwhile
‘
XQH                      ‘ Stop the program
```

See also:

BG, AB, KR, MS, IP

7.70 SR – Status Register

Purpose:

The “SR” Status Register is a read only parameter holding information on the current axis status.

Currently, “SR” should only be used to inquire the “In Target” bit condition of the axis.

The “In Target” status is indicated in bit #6 (“1” based, i.e. 0x20 Hex) of “SR”. For a complete description of In Target Status bit operation see the “TR” and “TT” parameters.

The “SR” parameter has the following attributes:

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	No.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	0 ÷ N.A.

Syntax:

```
XSR     ‘ Report value SR for X axis.
ASR     ‘ Report value SR for all axes.
```

Examples:

The following simple example demonstrates how initialize a PTP motion in X axis, then to wait for end of motion (monitoring “MS”) and “In Target” condition (monitoring “SR”).

```
XMO=1;XPS=0       ‘ Enables the motor and Set Position = “0”.
XMM=0;XSM=0       ‘ Set Normal Point To Point Motion Mode.
XAC=250000        ‘ Set Acceleration to “250,000”.
XDC=500000        ‘ Set Acceleration to “500,000”.
XSP=25000         ‘ Set Speed to “25,000”.
XAP=100000        ‘ Set Next PTP absolute location to “100,000”
TR=10;TT=160      ‘ Set Target Radius and Target Time
XBG               ‘ Start a Motion

@while (XMS != 0)         ‘ Wait for End Of Motion in X (XMS=0)
@endwhile

@while (XSR != 32)        ‘ Wait for In Target in X (XSR=32)
@endwhile
```

Another way to wait for “In Target” condition is to use the special “QW” command like in the following example:

```
$define  WaitForEndOfMotionX()     "XQW,100000"
$define  WaitForXInTR()            "XQW,101060"

XMO=1;XPS=0       ‘ Enables the motor and Set Position = “0”.
XMM=0;XSM=0       ‘ Set Normal Point To Point Motion Mode.
XAC=250000        ‘ Set Acceleration to “250,000”.
XDC=500000        ‘ Set Acceleration to “500,000”.
XSP=25000         ‘ Set Speed to “25,000”.
XAP=100000        ‘ Set Next PTP absolute location to “100,000”
TR=10;TT=160      ‘ Set Target Radius and Target Time
XBG               ‘ Start a Motion

WaitForEndOfMotionX()     ‘ Waits for End of Motion in X Axis
WaitForXInTR()            ‘  Waits for In Target in X Axis
```

See also:

MS, BG, TR, TT, QW

7.71 SV – Save Command

Purpose:

See “LD” Load Command.

7.72 TA – CAN Transmitting Address

Purpose:

This parameter sets the CAN Transmitting Address. The CAN transmitting address is the CAN address to which the controller responds to in any case a CAN message is received (the receiving address is defined in the “RA” parameter).

The CAN Transmitting Address must be saved to the flash memory, and the controller must be reset in order to change the CAN settings.

Changing RA/TA immediately re-initializes the CAN hardware to take the requested effect. Care should be taken, as changing RA/TA while working in CAN bus stops the communication with the PC. The parameters must still be saved to the Flash Memory (as in previous revisions) in order to be valid after boot.

A new error codes “EC_HW_INIT_ERROR=97” was added to indicate a CAN hardware initialization error.

In the FlexDC, in addition and independent to the standard RA and TA CAN addresses, listens and transmits on additional addresses. See the “ZI” keyword for more information.

Attributes:		
	Type:	Parameter.
	Axis related:	No.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	2.
	Range:	0 ÷ 2047.

Syntax:

```
XTA=2        ‘ Set CAN  TA=2.
XTA          ‘ Report value of TA.
```

See also:

CB, RA, ZI.

7.73 TC – Torque Command

Purpose:

The “TC” parameter keyword is used to set the Servo Analog Command value to a user specified value, when operating in open loop mode (when NC=1, and MO=1). In Open loop mode, the value of “TC” is directly forwarded to the controller Analog Command ***(Acmd)*** output, bypassing the PIV controller filter. Refer to the “NC” command for further information on how to enter open loop mode.

The value of “TC” is not saved to the Flash Memory. After MO=1 with NC=1, by default the analog output value commend is “0” to avoid motor motion (“TC” is set automatically to zero when MO=1).

It should be noted that in Open Loop mode the actual analog command is still limited by the “TL” (Torque Limit) parameter. Also, the control 2nd order filter may be used to monitor its operation and actual effect on the analog output value. The operation of the filter can of course be disabled by an appropriate flag (see 2nd order filter definitions in chapter 4, Part II). The user can choose to record the actual Driver Command (“PO”) value. In cases where the 2nd order filter is enabled, the actual value recorded is the step response of the filter. If no 2nd order filter is used, the actual value recorded is equal to the value commanded by “TC”. In any case the value is saturated by “TL”.

The value range of the “TC” parameter is 16 bit, reflecting the controller extended analog command resolution. This means that setting TC=32767 commands an analog command of +10 volts, while setting TC=-32767, commands an analog command of -10 volts. TC=0 of course sets analog command to 0 volts.

Note that the sign of the analog output can be inverted using the dedicated “CG” bits. The Analog offset can be set using the “DO” command.

The “TC” parameter has the following attributes:

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	-32,767 ÷ 32,767.

Syntax:

```
XTC=16384;      ‘ Reset value of  X axis TC to “16384”.
ATC=0           ‘ Set TC=0 for all axes (set analog Cmd =0).
```

Examples:

The following code example enables Open Loop mode on Y axis, and set the Y axis analog command output to +5 volts, and –10 volts:

```
YMO=0           ‘ Disables the motor before changing the NC.
YNC=1           ‘ Set NC=1 to indicate Open Loop for that axis.
YMO=1           ‘ Set MO=1 for Y Again.
YTL=32763       ‘ Set Command saturation to ± 10 Volts.
YTC=16384       ‘ Set command value to +50% (+5 Volts).

YTC=-32763      ‘ Set command value to -100% (-10 Volts).

YMO=0           ‘ Disables the motor before changing the NC.
YNC=0           ‘ Restore closed loop mode.
```

See also:

CG, NC, TL, 2nd order filter definitions, and Data Recording

7.74 TD – Timer Down

Purpose:

The “TD” parameter is an internal timer counting down towards “0”. The timer can be set to any value from 0 to 100,000,000. Upon reaching a count of 0 the timer stops.

“TD” should be used by user programs to generate delays or count times. The “TD” parameter is always reset to “0” after boot.

“TD” is an axis related parameter. There are 2 different internal timers that can be accessed by the user: XTD and YTD. There is no actual relation between XTD to the X axis. Each timer can be used by any program.

The “TD” timers count in the servo sample rate, i.e. 8,192 counts per second.

The “TD” parameter has the following attributes:

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	0 ÷ 100,000,000.

Syntax:

```
XTD=8192;       ‘ Set 1 Second Delay for X Timer.
XTD             ‘ Report the value of XTD.
BTD=8192        ‘ Set 1 Second Delay for both timers.
```

Examples:

The following code example sets the X Timer to 1 second delay and then waits for the timer to reach zero count. This is a simple way to implement a 1 second delay function.

```
XTD=8192            ‘ Set X Timer to 1 Second
@while (XTD > 0)    ‘ Waits for XTD to become zero
@endwhile
```

Another way to generate a 1 second delay is to use “TD” as above but then wait for “TD” to reach a zero value using the “QW” command:

```
$define TimerX          "XTD"
$define WaitTimerX()    "XQW,107000"

TimerX=16384            ‘ Set XTD=8192
WaitTimerX()            ‘ Waits for XTD to become zero
```

See also:

Refer to “Part III– FlexDC Macro Language”.

7.75 TL – Torque Limit (Analog Command Saturation)

Purpose:

The “TL” parameter limits the value of the analog output command to the servo amplifier. In applications where a current loop driver is used (most cases), the “TL” limit actually limits the motor current.

“TL” saturates the analog output command in both closed loop (NC=0) and open loop (NC=1,2,3) operation modes. See section 4, Part II, for the The Control Filter further information.

The range of “TL” is: 0 ÷ 32,767. TL=0 disables the analog command output to “0” volts. TL=32,767 is full range (100 % command), i.e.: ± 10 Volts.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	32,763.
	Range:	0 ÷ 32,767.

Syntax:

```
XTL=16384;      ‘ Set X Axis TL=16,384 (50% of Max Range).
ATL=32767       ‘ Set TL=32,767 in all axes (100 % limit).
```

Examples:

The following code example enables open loop mode on Y axis, but limits the Max analog command to ± 5 Volts.

```
YMO=0          ‘ Must Disables the motor before changing the NC.
YNC=1          ‘ Set NC=1 to indicate open loop for that axis.
YMO=1          ‘ Set MO=1 for Y Again.
YTL=16384      ‘ Set Command saturation to ± 5 Volts.
YTC=16384      ‘ Set command value to +50% (+5 Volts).
YTC=-32767     ‘ Set command value to -100% (-10 Volts), but TL actually limits the
                 actual outout value to –5 Volts.
```

See also:

NC, TC, IS

7.76 TR – Target Radius

Purpose:

The “TR” parameter defines the Target Radius in Encoder counts for the In Target detection logic. “TR” is used in conjunction with “TT” the Target Time and the Status register “SR” parameters.

During operation, while an axis is enabled (MO=1) and not in motion (MS=0), the real time control loop continuously checks the Position Error “PE”, and when ABS(PE) <= TR, for at least “TT” (Target Time) sample times, a dedicated bit in “SR” is set to high (logic “1’).

The In Target logic is usually used to let a host application (or a script program) to monitor the end of motion condition and wait for the axis to reach the desired target position within a specific defined error.

The “TR” parameter has the following attributes.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	2.
	Range:	0 ÷ 32,767.

Syntax:

```
XTR=10      ‘ Set X Target Radius to 10 counts.
ATR=20      ‘ Set All Axes Target Radius to 20 counts.
```

Examples:

See the “SR – Status Register” Command reference.

See also:

SR, TT

7.77 TT – Target Time

Purpose:

The “TT” parameter defines the Target Time in servo sample units for the In Target detection logic. “TT” is used in conjunction with “TR” the Target Radius and the Status register “SR” parameters.

During operation, while an axis is enabled (MO=1) and not in motion (MS=0), the real time control loop continuously checks the Position Error “PE”, and when ABS(PE) <= TR, for at least “TT” (Target Time) sample times, a dedicated bit in “SR” is set to high (logic “1’).

The In Target logic is usually used to let a host application (or a script program) to monitor the end of motion condition and wait for the axis to reach the desired target position within a specific defined error.

The “TT” parameter has the following attributes.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	2.
	Range:	0 ÷ 32,767.

Syntax:

```
XTT=160      ‘ Set X Target Time to 160 samples (20 msec in the FlexDC).
ATT=656      ‘ Set All Axes Target Time to 60 samples.
```

Examples:

See the “SR – Status Register” Command reference.

See also: SR, TR

7.78 VA / VD / VS – Vector Motion Parameters

Purpose:

The “VA” - Vector Acceleration, “VD” - Vector Deceleration, and “VS” - Vector Speed, are used for special X/Y Vector motions.

Vector motions are supported by the FlexDC for the execution of synchronized X and Y motions. The Vector motion syntax is fully compatible with the FlexDC syntax.

Vector motion is initialized in the FlexDC by issuing a BBG,-1 command (BBG command with a parameter equals to –1).

When a BBG,-1 command is executed, the controller first computes the Vector Distance and Vector Angle, based on the X and Y motion distance components.

The vector distance is not directly defined along the vector but instead it is defined as its X, Y components. The desired motion distances for the X and the Y axes are defined normally using AP (or RP). The desired distance along the X axis is (XAP-XDP) and (YAP-YDP) for the Y axis. The DP value represents the desired current position (before the motion) while AP is the desired target position. The Vector Distance and Angle are computed as follows:

Vector Distance = SQRT { $(XAP-XDP)^2$+ $(YAP-YDP)^2$ }

Vector Angle = ATAN { (YAP-YDP) / (XAP-XDP) }

Once the Vector Angle is determined, it is used to compute the accelerations, decelerations and speeds projection on both X and Y axes, as follows:

XAC = AVA * SIN (Vector Angle)

XDC = AVD * SIN (Vector Angle)

XDL = AVL * SIN (Vector Angle)

XSP = ASP * SIN (Vector Angle)

YAC = AVA * COS (Vector Angle)

YDC = AVD * COS (Vector Angle)

YDL = AVL * COS (Vector Angle)

YSP = ASP * COS (Vector Angle)

In the next phase (of the BBG,-1) command, both the X and Y axes are commanded for synchronized motion, based on the AC/DC/DL/SP parameters computed above. The actual axis specific AC/DC/DL/SP are being overwritten by the BBG,-1.

Note that in this case, the BBG,-1 command must be use BBG. XBG,-1 or YBG,-1 starts a motion in X or Y only, with unexpected motion parameters, and should be avoided. The BBG,-1 function calculates new values to the above parameters independently of the Motion Mode and the Special Mode parameters. As a result, a vector motion is created for all motion modes which use the above parameters, including: Jogging, Point-To-Point, Repetitive Point-To-Point etc.

Motion modes which do not use the above parameters (such as ECAM) is not affected by the -1 parameter (BG,-1). However, the above parameters in any case are recalculated and overwritten.

The user can still modify all parameters which supports on-the-fly modifications (such as SP). However, it affects each axis independently and causes a motion that may not be consistent with the originally desired vector. VA, VD, VL and VS can still be modified on the fly, but will not affect since these parameters are used only for the pre-calculation within the BBG,-1 command function.

Note that in the current implementation, the controller does not "remember" that it is in vector (common) motion. The BBG,-1 performs a pre-calculation which prepares the SP, AC DC and DL parameters of both axes for a synchronized motion along the vector and initiate a motion for both axes. From this point the two axes performs normal independent motion according to their MM and SM parameters.

While this is a very simple and predictable behavior, it has a disadvantage that the axes are not truly linked together. For example, a fault in one axis will not affect the other.

Attributes and Syntax:

Except from being non-axis related parameters, the “VA” - Vector Acceleration, “VD” - Vector Deceleration and “VS” - Vector Speed, are analogues to the axis specific parameters “AC” / “DC” and “SP”, and has similar attributes.

Examples:

The following example starts a common XY vector motion:

```
BMO=1;BMM=0;BSM=0
BVS=100000;BVA=1000000;BVD=1000000
XAP=50000;YAP=600000
BBG,-1
```

Se also:

AC, DC, DL, SP, MM

7.79 VR – Get Version Command

Purpose:

The “VR” command retrieves the controller Firmware and FPGA versions. The standard Version Command response report (in RS232 communication protocol) has the following syntax:

”**FlexDC Motion Controller 2,101,2,5,16**”. The following interpretation is applicable:

FlexDC Motion Controller 2: Indicates the new FlexDC product code.

101: Indicates that Firmware Version 1.01 is installed.

2: Indicates that this is a 2 axis version.

5: Indicates the FPGA version (5).

16: Indicates the Macro Buffer size in kBytes (16 kBytes).

Note: Firmware version must comply with FPGA versions. Downloading firmware versions without prior authorization from Nanomotion Ltd. is not allowed, and might result in a malfunctioning (un-expected results) board.

In CAN bus communication, the standard VR report has the following syntax (see Table 31):

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Type Number	FW Ver Hi	FW Ver Low	Num Axes	FPGA Version	Reserved	Macro Size Hi	Macro Size Low
=52	=0	=101	=2	=5	0	=0	=16

Table 31: FlexDC to Host - CAN VR Version Report Message Format

The VR (Version Report) command also supports receiving a parameter as part of the command syntax. Calling “VR” without any parameter is fully compatible to previous revisions version report format (indicated above). However, the controller now also supports the following additional version reports:

“BVR,1” : Reports Boot and Single or Dual Axes Controller Version.

“BVR,2” : Reports Firmware (Major and Minor) Versions, with its release Date and Time.

“BVR,3” : Reports the FPGA Version.

In current firmware version, special VR requests are supported on the FlexDC in RS232 only.

The “VR” command has the following attributes:

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Command Allows Parameter:	Yes (see above).
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

XVR	‘ Standard Version Report
XVR,2	‘ Reports Firmware (Major and Minor) Versions, with its release Date and Time (RS232 only)

Examples:

See Syntax Above.

7.80 WT – Wait Period

Purpose:

This parameter sets the Wait time for Repetitive Point to Point. When the controller is in MM=0 (PTP) and SM=1, the motion is repetitive. This means that the axis is commanded to perform a PTP motion to the specified absolute position and then, after the motion is completed and a user specified delay (WT) is completed, a new motion is automatically initiated to the starting position (AP is updated to this value). This back-and-forth motion is repeated until stopped by one of the following clauses: AB (abort), ST (stop), KR (Kill repetitive), and MO=0.

The WT parameter defines the delay time in number of servo samples (each is approximately 122 [us] or 1/8,192 of a sec) between the back-and-forth motions.

Attributes:		
	Type:	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	0 ÷ 800,000,000.

Syntax:

```
XWT=0;          ‘ Set X Axis WT=0.
AWT=16384       ‘ Set WT= 16384  in all axes.
```

Examples:

The example shows starting a Repetitive motion in X axis from Position “0” to Position “100,000” using 1 sec “WT” Wait between the motions.

```
XMO=1;XPS=0              ‘ Enables the motor and Set Position = “0”.
XMM=0;XSM=1              ‘ Set Repetitive Point To Point Motion Mode.
XAP=100000               ‘ Set Next PTP absolute location to “100,000” counts.
XAC=100000;XDC=100000    ‘ Set Acceleration to “250,000”.
XSP=25000                ‘ Set Speed to “25,000”.
XWT=16384                ‘ Set 2 second delay between motions.
XBG                      ‘ Start a Motion
```

See also:

AC, DL, SP, MM, BG

7.81 WW – Profiler Smooth Factor

The FlexDC supports an advanced, symmetric S-curve like profile smoothing algorithm. The smoothing is controlled by the “WW” parameter.

“WW” can be set to 0 to avoid any profile smoothing. In that case the generated position velocity profile is pure trapezoidal (or triangular). If “WW” is set to 12, the smoothing is set to its maximal value. In this case the generated profile has full smoothing, and the velocity trajectory is not purely trapezoidal.

The “WW” parameter is used by the controller as a power of 2 coefficient for the smoothing time value. For example, WW=6 means that smoothing is done over a period of time of 2^6 sample time, i.e. 8 msec. The resulted profile generates its full acceleration value in the 2^6 sample time.

Setting WW=12 to its Maximal smoothing value of 2^12, results in a 0.5 sec acceleration smooth period.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	Not In Motion.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	0 ÷ 12.

Syntax:

```
XWW=0;      ‘ Set XWW=0 (No Smoothing for X Axis)
AWW=8       ‘ Set WW= 8 for all axes.
```

See also:

MM

7.82 XC – Last Capture Position Latch

Purpose:

The “XC” parameter is used in conjunction with the Capture function to report the last captured position of an axis.

The last Captured location is stored by the controller firmware in the “XC” parameter for each axis independently (i.e.: XXC, YXC axes respectively). The user should note that when “PS” is updated, the value of “XC” becomes meaningless. The Capture feature implementation does not support hardware or software buffers. Whenever a Capture is detected, the last value of “XC” is overwritten and lost. As indicated above, “XC” is an axis related parameter keyword. Each axis holds its own Captured Position Location value.

“XC” has the following attributes:

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	No.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	- 2,147,000,000 ÷ 2,147,000,000.

Syntax:

XXC ‘ Reports the Last Captured position of X axis.

YXC ‘ Reports the Last Captured position of Y axis.

BXC} ‘Pushes the last X and Y Captured positions to the Stack top.

See also:

XN

7.83 XN – Capture Events Counter

Purpose:

The “XN” parameter is used in conjunction with the Capture function to report the number of Capture Events.

Each time the hardware Captures (Latches) a new position, the total number of Capture events (“XN”) is incremented accordingly. The user can reset this variable to “0”, and monitor its value to wait for a Capture event within a script program. This can be used for example to signal events to a host computer whenever a Capture event is sensed.

“XN” is an axis related parameter keyword. Each axis holds its own Capture index counter.

“XN” has the following attributes:

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Command Allows Parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	0.

Syntax:

```
XXN      ‘ Reports the X axis Capture Events Number.
YXN=0    ‘ Reset Y axis Capture Event counter.
```

See also:

XC

7.84 ZI – CAN Array

Purpose:

"ZI" holds various CAN related parameters. These parameters are used in numerous cases:

1. CAN remote unit addresses.
2. EDB modes.
3. Additional transmit and receive addresses.

For CAN remote unit addresses, "ZI" is usually used in script programs in order to define the remote unit's addresses.

The "ZI" array is an axis related array, with size of 2x12 elements. Each element in the array is a LONG format number, which can be assigned, with any value at any time. The index range of the "ZI" array is: 1 ÷ 12 (see Table 32).

	X	Y
1	Remote Transmit Address (The remote message is sent from the FlexDC on this address) for macro 'X'.	Remote Transmit Address (The remote message is sent from the FlexDC on this address) for macro 'Y'.
2	Remote Receive Address (The remote message is received in the FlexDC on this address) for macro 'X'.	Remote Receive Address (The remote message is received in the FlexDC on this address) for macro 'Y'.
3	Additional CAN TA Address	
4	Additional CAN RA Address	EDB Configuration
5	EDB Error Status	EDB Receiving CAN Address
6	Buffer1 Array Code	Buffer2 Array Code
7	Buffer1 Axis Code	Buffer2 Axis Code
8	Buffer1 Current Index	Buffer2 Current Index
9	Buffer1 Increment Value	Buffer2 Increment Value
10		
11		
12		

Table 32: "ZI" array

The ZI keyword has the following attributes:

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	Yes, size = [2][12].
	Assignment:	Yes.
	Command Allows parameter:	---.
	Scope:	All.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	- 100,000 ÷100,000.

8 Communication and Program Error Codes

The following table lists ALL possible communication and program Error Codes (EC) supported by the FlexDC firmware. The error codes listed below are applicable to both communication errors as well as program execution error codes.

The errors in Table 33 are applicable in both single and double axes versions of the FlexDC, unless specified otherwise.

For program related error codes, refer to "Part III– FlexDC Macro Language" for more information.

EC Val	EC Code Name	Error Description
0	COM_OK	No Error in history. This value is received also by resetting the EC variable.
1	BAD_KEYWORD_AXIS	This error is issued in the following cases: Bad axis prefix was used for the command / parameter. Keyword does not support one of the axes in the group prefix used. e.g.Group 1 is set to 1023, and the AAC clause is sent.
2	SYNTAX_ERROR	This error is issued once a wrong syntax for current clause was sent to the controller. It may be one of the following: Unrecognized operator. Unrecognized Keyword. Macro label not according to syntax defined. General clause not according to the defined syntax.
3	BAD_ARRAY_INDEX	This error is issued if a clause includes an array variable, and the index of the array variable is not in the correct range for the specific array. Note, ALL start array indexes are 1 AND NOT 0.
4	NOT_IN_SCOPE	Not used.
5	BAD_NUMBER_RANGE	This Error Code is issued if an assignment clause is not In the range for the specific variable.
6	READ_ONLY	This Error Code is issued if an assignment clause was issued on a Read Only keyword.
7	NOT_AN_ARRAY	This Error Code is issued if a clause consists of array referral on a keyword that is NOT an array. This is detected in the following matter:

EC Val	EC Code Name	Error Description
		An Open Parenthesis appears in the clause, after the Keyword. (Or at all…). Digits appear immediately after the keyword in a clause.
8	AN_ARRAY	Not used
9	RECEIVE_BUF_FULL	This Error Code is issued if the string sent by RS232 exceeds the maximum of 128 characters.
10	TOO_LONG_MESSAGE	Not used.
11	FAIL_ERASE_FLASH	Not used.
12	FAIL_WRITE_FLASH	Not used.
13	TOO_LARGE_NUMBER	This Error Code is issued if the clause being interpreted includes an assign command with more than 12 digits.
14	WRONG_MOTION_PARAM	This Error Code is issued if during a motion the controller detects wrong motion parameters. This can happen for example if during ECAM motion, bad ECAM data or wrong motion of the master axis is detected. **Note:** the ECAM motion mode is not supported in the current FlexDC firmware.
15	WRONG_RECORD_PARAM	This Error Code is issued if the number of recording points exceeds the size of the DA array.
16	STILL_RECORDING	This Error Code is issued when the user sends a Begin Recording Command (“BG”, or “BG,1”) while Data Recording is still in process, i.e. RR > 0. Data Recording can be started only when previous recording session was terminated. Note that the controller does not check if previous buffers were uploaded or not. Issuing a Begin Recording command always overrides old data.
17	NO_RECORDING_DATA	Not used.
18	CAN_BAD_ARRAY_CODE	Not used.
19	CAN_BAD_LENGTH	Not used.
20	STACK_FULL	This Error Code is issued once a macro source clause included a push command once the stack was full. The Push command can be to one of the following stacks: Number Stack – As a result of a number push command. The number stack size is 15. Program Stack – As a result of a ‘Call’ command. The Program’s Stack size is 15.
21	STACK_EMPTY	This Error Code is issued once a macro source clause included a pop command once the stack was empty. The pop command can be to one of the

EC Val	EC Code Name	Error Description
		following stacks: Number Stack – As a result of a number pop command. Program Stack – As a result of a 'Return' command.
22	NOT_ENOUGH_ARG	Not used.
23	DIVIDE_BY_ZERO	This Error Code is issued once a macro source clause included a division operator to a denominator with the value of zero.
24	BAD_CONSTANT	Not used.
25	NO_VALID_MACRO	This Error Code is issued once the 'QI' keyword is issued without any macro in the system.
26	CAN_NOT_FIND_LABEL	This Error Code is issued once the QE,#LABEL clause is sent with an un-existing label. This means the user issued an execute command to a specific macro routine, but the specific routine does not exist.
27	BAD_NUMERIC_FIELD	Not used.
28	CLAUSE_TOO_LONG	This Error Code is issued once a clause exceeds the length of 255.
29	MACRO_END	This Error Code is issued once a macro One Step or an macro Execute command were issued and the macro reached the end.
30	MACRO_POINTER	This Error Code is issued once the macro pointer is not in the limits of the macro code or one of the following functions lack a pointer as a parameter: Jump. Call.
31	TARGET_ADDRESS	Not used.
32	TOO_LONG_LABEL	This Error Code is issued if a label parameter exceeds the limit of 12. It can happen in one of the following functions: 1. Jump. 2. Call. 3. Execute.
33	PARAM_NOT_ALLOWED	This Error Code is issued if a parameter in a dedicated clause, exceeds the limits for this parameter.
34	PARAM_OUT_OF_RANGE	This Error Code is issued when a command is given a parameter out of range, or when trying to assign a parameter with a value out of its range. Please check the relevant command or parameter keyword reference for more information about the allowed range for the specific parameter.

EC Val	EC Code Name	Error Description
35	STATE_NUMBER_RANGE	Not used.
36	CAN_BAD_SPECIAL	This Error Code is issued when a CAN message size, in the special download array feature, differs from 1, 4 or 8.
37	CAN_REMOTE_TIMEOUT	The FlexDC has the ability to send, via macro, CAN strings to remote units. If the remote unit does not reply within a given timeout (1 second), this error is issued.
38	PARAM_EXPECTED	This error is issued when a command requiring a parameter is issued with out one. Please check the relevant command keyword reference for more information about the command's parameter.
39	BAD_PARAM_TYPE	Not used.
40	BAD_PARAM_LENGTH	Not used.
41	CAN_NOT_ACTIVE	Not used.
42	BAD_ARGUMENT	Not used.
43	CAPTURE_DISABLED	Not used.
44	BAD_PARAM_SYNTAX	This error is issued when the wrong parameter is attached to a clause.
45	ARRAY_PARAM	Not used.
46	DOWNLOAD_OVERFLOW	This error is issued in the following cases: The macro buffer, during the download macro, exceeds the maximum macro size. An index overflow occurred in the special CAN download buffer sequence.
47	NEEDS_MOTOR_ON	This error is issued if the condition for the specific clause is having the motor ON, while the clause was issued when motor is OFF.
48	NEEDS_MOTOR_OFF	This error is issued if the condition for the specific clause is having the motor OFF, while motor is ON.
49	NEEDS_MOTION_ON	This error is issued if the condition for the specific clause is having the motion ON, while motion is OFF.
50	NEEDS_MOTION_OFF	This error is issued if the condition for the specific clause is having the motion OFF, while motion is ON.
51	MACRO_NOT_INITIALIZED	This error is issued if a macro related clause is initiated, before the macro was initialized.
52	NEEDS_COMMUNICATION	This error is issued if a clause was not sent via communication whilst the clause was defined to be of KW_SOURCE_MUST_BE_COM.
53	SW_LIMIT_ERROR	A Point-To-Point motion was initiated into one of the

EC Val	EC Code Name	Error Description
		software limits.
54	AXIS_NOT_SUPPORTED	The Assignment to MO=1 failed, as the current hardware does not support the axis set.
55	UNSUPPORTED_MODE	The method called is no longer supported in the current firmware version.
56	EC_UNSUPPORTED_DRIVER	Set when a CG for Un-Supported Driver type is configured in CG in FlexDC.
60	PG_ERR_MODE_PARAM_NOT_VALID	This error is issued by “PQ,1” (Enable Compare Function) if the requested Compare Mode defined by PG[i][1] is out of its range. In the current firmware version only Modes 0 and 2 are supported for Compare Function).
61	PG_ERR_PULSE_MODE_PARAM_NOT_VALID	This error is issued by “PQ,1” (Enable Compare Function) if the Pulse Width Mode Parameter defined by PG[i][6] is out of its range. The allowed range for the Pulse Width Mode Parameter is: “0” or “1”
62	PG_ERR_PULSE_WIDTH_PARAM_NOT_VALID	This error is issued by “PQ,1” (Enable Compare Function) if the Pulse Width Parameter defined by PG[i][5] is out of its range. The allowed range for the Pulse Width Parameter is: “0” to “3”.
63	PG_ERR_PULSE_POL_PARAM_NOT_VALID	This error is issued by “PQ,1” (Enable Compare Function) if the Pulse Polarity Parameter defined by PG[i][7] is out of its range. The allowed range for the Pulse Polarity Parameter is: “0” or “1”.
64	PG_ERR_PD_PARAM_NOT_VALID	This error is issued by “PQ,1” (Enable Compare Function) if the Distance Parameter defined by PG[i][2] is out of its range. Out of range values for Distance are: 0 in all modes. Out of +/-32,767 range in Mode 0. Not equal +1 or –1 in Modes 2 and 3.
65	PG_ERR_PS_PE_PARAM_NOT_VALID	This error is issued by “PQ,1” (Enable Compare Function) if the Start Point or End Point Parameters defined by PG[i][3] and PG[i][4] are not valid. These parameters are validated only in Modes 2 and 3 (see specific operation mode description for more details about limitations on PStart and PEnd).
70	QW_AXIS_OUT_OF_RANGE	This error is issued if the **Axis**, which is derived from the parameter to the QW command, is out of range.
71	QW_CODE_OUT_OF_RANGE	This error is issued if the **Code**, which is derived from the parameter to the QW command, is out of range.
72	QW_BIT_OUT_OF_RANGE	This error is issued if the **Bit**, which is derived from

EC Val	EC Code Name	Error Description
		the parameter to the QW command, is out of range.
73	QW_LOGIC_OUT_OF_RANGE	This error is issued if the **Logic**, which is derived from the parameter to the QW command, is out of range.
74	QW_INTERNAL	This error is issued if an internal error, due to the QW command occurred.
97	HW_INIT_ERROR	An error occurred trying to re-initialize the CAN bus via re-setting the RA or TA variables
98	FLASH_ERASE	This Error Code is issued if an error occurred during the Saving to Flash Procedure. The error is related to erasing the Flash Memory.
99	FLASH_VOLTAGE	This Error Code is issued if an error occurred during the Saving to Flash Procedure. The error is related to the flash voltage.
100	FLASH_ACK_TIMEOUT	This Error Code is issued if an error occurred during the Saving to Flash Procedure. The error is related to acknowledge time out from the flash hardware.
101	FLASH_SUSPEND	This Error Code is issued if an error occurred during the Saving to Flash Procedure. The error is related to flash suspend.
102	FLASH_WRITE	This Error Code is issued if an error occurred during the Saving to Flash Procedure. The error is related to flash write.

Table 33: Communication and Program Error Codes

Part III – FlexDC Macro Language

Script Programming Language and the Integrated Development Environment

9 Introduction

Part III describes the FlexDC macro-programming language and environment.

In order to benefit from the best performance possible from the FlexDC controller hardware platforms, the macro environment is designed with a simple fast-execution engine, and a powerful programming development and debugging environment. The embedded FlexDC and its firmware are responsible for the real time macro execution, while the PC based Nanomotion Shell Application provides the environment for editing, pre-compiling, compiling, downloading and debugging macro programs.

To support this structure, the macro programs' logic combines a reduced set of low level commands supported by the macro execution engine, and a wider, more enhanced set, of programming commands that are supported by the Integrated Development Environment. While the low-level macro execution engine (embedded FlexDC firmware) supports only the necessary numbers stack, stack operations and basic flow control commands, the higher level Nanomotion Shell Application environment supports advanced conditions (If, While, etc.) and advanced expression calculations. Translation from the high to low level language is fully automatic, and is under operated by the Nanomotion Shell Pre-compiler module (integrated into the Nanomotion Shell Application program).

The only reasonable limitation that this programming structure imposes is that for the development of a macro program you should have a Windows based PC platform available. This is if course required only for the development and debugging stage. Once the program was downloaded to the embedded hardware and saved to its Flash Memory, the controller program is fully autonomous and supports stand alone operation.

The benefits from this structure include a wider language base, faster real-time execution and a powerful Debugging Environment as well as other benefits that are outlined in Part III.

10 FlexDC Macro Engine

10.1 General FlexDC Macro Program Structure

The FlexDC controller supports two macro programs: X and Y macros. Both macro programs share the same macro source code (or buffer), as described in Figure 12:

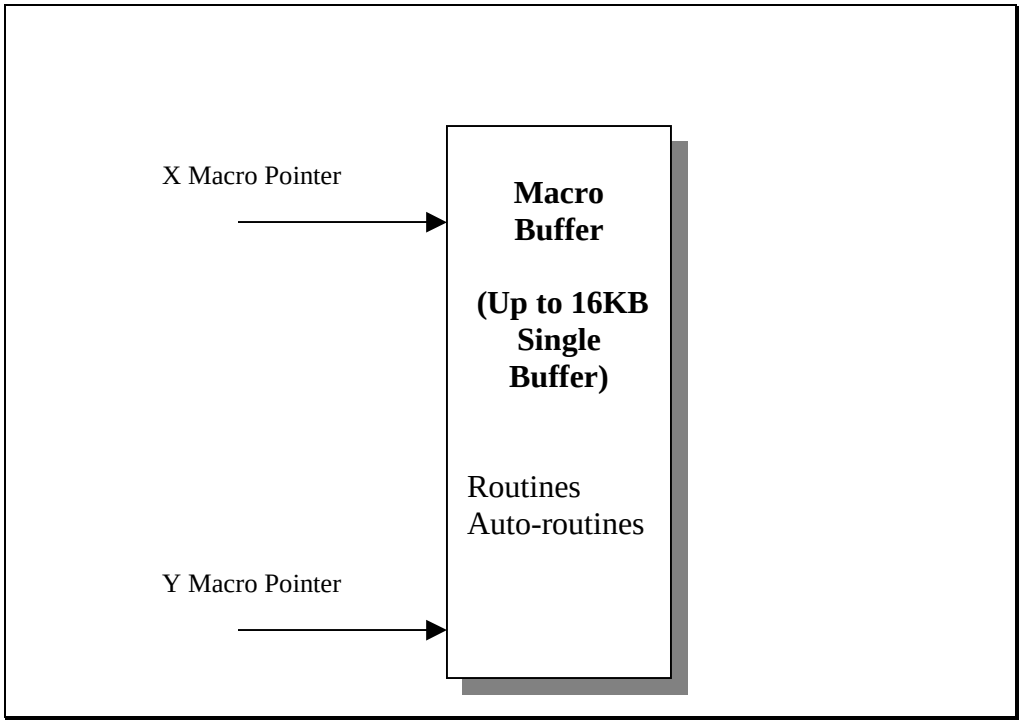


Figure 12: Macros and Macro Source Buffer

Careful writing of the macro (for example different routines) enables independent macro programs. However, some routines can be shared if required.

In addition, both macro programs share the same variables and have the same access to all the controller's commands and parameters. It is thus not necessary that the X macro will handle X motions while the Y macro will handle the Y motions. A quite more logical approach is that one of the programs (lets says X) will handle all motions and one will handle all I/Os logic (as a PLC).

The macro programs are executed in parallel (no priority logic): one clause from the X macro and then one clause from the Y macro.

The two macro programs are completely independent. The user can execute the macros in parallel, or stop each one at any time. Independent automatic routines are also assigned to each macro program.

10.2 External Communication vs. Macro Execution Priority

Communication clauses have higher priorities over the program execution. Program clause is executed only if there is no communication (RS232 or CAN) clause waiting. It is thus clear that the communication load influences the macro execution speed.

10.3 Macro Handling Keywords

In order to support execution and handling of macro programs within the FlexDC, a set of keywords was dedicated in order to support macro programming, see Table 34.

Note on commands syntax: like all other commands of the FlexDC, the keywords **should be preceded with “X” or “Y” to identify the respective macro.**

Keyword	Description
QB[ ]	A macro related array of 20 breakpoints pointers (-1 to disable a pointer and following pointers). Should not be used.
QC	Reports the last macro runtime error (if there was any).
QD	Downloads a macro
QE	Execute macro from the current macro pointer (QP)
QH	Halt macro execution
QI	Initialize macro and its internal variables
QK	Kill macro execution (also stops all motions of both axes)
QL	Loads the macro from the Flash Memory. Automatically after power ON or reset. This command is currently not implemented. Using the LD (for loading parameters) also loads the macro.
QN	Displays the macro stack
QP	Holds the current macro pointer
QQ	Uploads the program stack (queue of return addresses)
QR	Reports the macro status
QS	Saves the macro to the Flash Memory.
QT	Execute single macro clause (trace) from the current macro pointer (QP)
QU	Uploads a macro
QV	Uploads all macro descriptive data
QZ	Clears all the numbers stack

Table 34: FlexDC Macro Program Handling Keywords

Although most of the keywords described above are generally used from an external communication line, some commands can also be included from within a macro code. For example, the QE and QH (XQE/XQH or YQE/YQH) keywords may be used from an X or Y macro routines to start and stop the execution of the second macro program.

For further information refer to chapter 15, Part III, (see sections 15.3 and 15.4).

10.4 Low-Level Expressions Handling and the Numbers Stack

Almost any macro application involves expressions. Expressions are used to perform calculations. The standard FlexDC language syntax supports only parameters report and assignment, such as:

XSP=10000

XAR[34]=5

More complex expressions are not supported by the low-level FlexDC macro language, for example:

XSP=XAR[34]

XSP=XAR[34]+10000

XSP=XAR[34]*XAR[567]+20000*XAR[899]

In order to support variety of expiration types, as well as relation expressions that are necessary for conditional program flow, each macro - X and Y has its own numbers stack. Dedicated keywords are added to support pushing and popping to/from the stack and mathematical, as well as relational operator keywords are added.

These numbers stacks (to distinguish from the internal program stack that is used to store return pointers for CS commands) are currently accessible also from the RS232 communication line, mainly for debugging. This way, the macro can perform any expression (without complexity limitations and without operators preceding limitations). Dedicated macro keywords, such as the JT (Jump if True), CT (Call subroutine if True) automatically uses the last stack element as their input arguments. This structure is similar in concept to any μ-controller assembly language syntax.

Below are some examples demonstrating this simple concept.

Example #1:

- *A possible standard expression such as:*

'JP#ABCD,XPS>10000',

which means: Jump to label #ABCD if X position is greater than 10000 will be written in macro syntax as follows:

'XPS};10000};>;JT#ABCD',

which means: push XPS, push 10000, pop the last two elements and push 1 if greater, 0 if smaller. Pop from stack and jump if 1. The stack remains empty at the end of this process, as it should be.

Example #2:

- *Another simple examples is the following assignment:*

'XSP=XAR[34]+10000',

which is implemented in the macro syntax as follows:

'XAR34};10000};+;XSP{',

which means: push XAR[34], push 10000, sum stack top two elements (pop twice and push summation result to the stack top automatically), then pop the result to XSP.

Also here the stack remains empty at the end of this process.

Example #3:

- *A more complex expression such as:*

'XSP=XAR[34]*XAR[567]+20000*XAR[899]',

- *Is implemented in FlexDC macro syntax as follows:*

'XAR34};XAR567};*;XAR899};20000};*;+;XSP{',

Which means: push XAR[34], push XAR[567], multiply stack top two elements (pops twice and push multiplication result to the stack top automatically), push XAR[899], push 20000, multiply stack top two elements again (pops twice and push multiplication result to the stack top automatically), sum the stack top two elements (pop twice and push the summation result, and finally pop the result to XSP.

Also here the stack remains empty at the end of this process.

Again, as noted above, the main advantage of this parsing syntax is that expression complexity is practically unlimited, and does not required complex run-time parsing mechanisms to be implemented by the μ-controller macro engine.

Nevertheless, for those users who are not used to programming in this way, the PC based pre-compiler supports standard expression parsing (translating normal expressions to the low-level syntax). For full description of the pre-compiler support, refer to chapter 13, Part III.

Additional keywords are added to support expressions and the numbers stack, see Table 35. These keywords are usually used from within a macro program, but are also supported from the communication, mainly for debugging purposes.

Keyword	Description
QN	Displays the macro stack
QZ	Clears all the numbers stack
}	Push (without argument, duplicates last stack element)
{	Pop (without argument – remove last stack element)
+	Add
-	Subtract
*****	Multiply
/	Divide
||	ABS
+-	Negate
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
!	Logical NOT (result is always 0 or 1)
>0	Is positive
<0	Is negative
=0	Is zero
!0	Is not zero

Keyword	Description
>	Is greater
<	Is smaller
==	Is equal
!=	Is not equal
>=	Is greater equal
<=	Is smaller equal

Table 35: Macro Program Operators

For further information refer to chapter 15, Part III (see sections 15.3 and 15.4).

10.5 Variables and Indirect Addressing

10.5.1 Variables

Almost any macro application needs variables to hold temporary values, to perform calculations, and to transfer parameters to its subroutines and between the host application (if exists) and the macro.

Since the low-level programming syntax does not support dynamic variables allocation (or free naming), several types of arrays are supported by the FlexDC language, and can be freely used from within a macro program as temporary variables (registers).

Another common requirement is to perform indirect addressing to an array of variables. This is to enable based-indexing to a set of parameters. In order to support this feature, a special array keyword is defined. This is the IAi array (see description below).

For ease of use, each array described below may be accessed through both the communication lines and internally, from within a macro, with or without the square index brackets [ ].

Square brackets must be used when using indirect addressing, as an exception, i.e. AR[XIA6] (see further explanation below).

The following arrays are currently defined:

Array Name	Axis Related	Size in FlexDC	Notes
AR[i]	No	1 * 16000 elements, (i.e. AR1 through AR16000)	This is a general purpose array. And may be used freely from within a macro program. This array is also used for Compare Mode and For Data Recording. The data recording is performed from index 16000 backwards. Therefore, once performing data recording and using the AR array, the user MUST act carefully.
PA[i]	Yes	2 x 100 elements. (i.e., XPA1 through XPA200, and YPA1 through YPA200).	This is a general purpose array (Parameters Array), intentionally defined for temporary usage of macro variables. No other internal controller function uses this array under any circumstances.
IA[i]	No	1 x 50 elements. (i.e., XIA1 through XIA50).	This is a general purpose index array (Index Array), intentionally defined for temporary usage of macro variables, and indirect addressing (see explanation below). No other internal controller function uses this array under any circumstances.

Note:

- *When working with non-axis related arrays (i.e. AR[i] and IA[i]), using either X, Y or group prefixes, as a preceding character, yields access to the same internal register (this is true of course for all non-axis related commands supported by the FlexDC syntax). The selection of which preceding character to use is user free. With the multi dimensional PA[i] array, one example may be to use XPA for storing X motion parameters and YPA to store Y motion parameters etc. This decision is again completely left for the macro programmer to decide.*

10.5.2 Indirect Addressing

All arrays, described above, support indirect addressing. Indirect addressing is required to enable based-indexing to a set of parameters.

For example, a single homing routine that should serve more then one axis (with each axis having different motion parameters) may use indirect addressing accessing the global parameters array. The calling task should store the start index of the relevant axis parameters in an IA array element, and the homing routine accesses the parameters array using the IA index. Using this method a lot of conditions and program space may be saved. Another example is management of stack pointer for recording of user specified data.

Indirect addressing is supported by using the IA array as an index array.

Example:

- *Assignment of constant value to AR[56] can be written simply by:*

'XAR56=1234', or using indirect addressing by:

'XIA4=56'

'XAR[XIA4]=1234'

First IA4 is assigned with the address '56', and then XIA4 is used as an indirect index to AR.

Notes:

- *When IA is used as an indirect address, the square brackets [ ] must be used. There is no need in this case for preceding characters (X or Y) within the square brackets, before the IA.*
- *Only the IA array may be used for indirect addressing, but all other arrays support the use of IA as an indirect address (including IA itself).*
- *The IAi array elements may also be used within any normal expression, and are not only limited to indirect addressing.*

10.6 Labels and Subroutines Names

Labels are required for two main tasks: (1) To define subroutine names (starting location), and (2) To define locations to jump to (for program flow control).

Labels are defined in the FlexDC programming language as follows: "#LABEL".

The "#" sign must precede any definition of label. The "#" must be the first character in a line.

Followed by the "#" sign is the label definition. Labels may include ASCII printable characters with no blanks. Maximal label size is 12 characters. (see restrictions on labels definitions below).

Ending a label definition is the "**:**" sign.

Note:

- *To speed up real time execution of the macro program, the pre-compiler module translates label definitions to internal pointer locations. This eliminates the need to interpret the labels at run-time execution. Nevertheless, this process is completely transparent to the user. The user uses the actual ASCII labels, as defined in the macro source code for external and internal routine calls, as well as for the various jump functions. For further information about label and pointer definitions see section 12.4, Part III.*

Example:

- *If user's homing routine is defined with the following label:*

#HOME_X:

the following communication command will start execution of this subroutine:

XQE,#HOME_X

From within a macro procedure, the same homing function will be called using the "Call Subroutine" function (see further description on section 10.7, Part III, regarding macro flow control):

XCS,#HOME_X

10.6.1 Restrictions on Labels Definition

The "#" sign must be the first character in a label definition line.

Maximal label size is limited to 12 characters (not including the proceeding "#" and the terminating"**:**"). Labels that are called by the PC (or any other host) are restricted to a length of maximum 6 characters.

Labels may include ASCII printable, alphanumeric characters only. Labels are case sensitive. The underscore "_" character may be used within a label definition, but can not start a label definition. No blanks are allowed within a label definition.

10.6.2 Ending a Label Definition is the ‘:’ Sign

In order to implement high-level program flow statements such as "if" and "while", the pre-compiler module automatically generates internal program labels. The following labels are saved keywords and should not be used by the user: ‘SI_<CONST>:’, ‘EI_<CONST>:’, ‘WH_<CONST>:’, ‘EW_<CONST>:’, ‘UF_<CONST>:’, ‘CF_<CONST>:’, ‘EF_<CONST>:’, where CONST stands for a constant label index number (1,2 ..).

Labels that are defined with an additional ‘#’ prefix, i.e. ##WAIT_INPUT:, are not downloaded to the controller, and can not be called by the PC (or any other host). The additional "#", is calculated as an additional character, in the maximum label length count.

Examples for valid label definitions:

#MAIN:

#HOME_X:

#L_1:

#L__p1:

#L__p2:

Examples for non valid label definitions:

```
#MAIN                ‘ No terminating ‘:’.
#_LS:                ‘ Label can not start with ‘_’.
#HOMING_X_AXIS:      ‘ Label too long.
#SI_1:               ‘ Saved keyword.
```

10.7 Macro Flow Control

Any programming language should support program flow control commands, to allow controlling the program flow during run-time. Flow control commands implement functions such as calling a subroutine (Call Sub), jumping to a certain location in the program (Jump to a specific label or to a pointer location), conditional jumps, etc.

The table below describes flow control commands supported by the FlexDC low level macro engine. For further information see chapter 15, Part III.

Keyword	Description
CS	Call subroutine at a new macro pointer
CT	Call subroutine if last stack element is TRUE (not zero)
CF	Call subroutine if last stack element is FASLE (zero)
JP	Jump to a new macro pointer
JT	Jump if last stack element is TRUE (not zero)
JF	Jump is last stack element is FALSE (zero)
JZ	Jump to a new macro pointer and clear subroutines stack (to restart the macro with subroutines stack clear)
RT	Return from a subroutine

Table 36: Macro Program Flow Control Keywords

Note that the low-level macro engines support a limited number of flow control commands, namely Calls, Jumps and Return. Conditional calls and or jumps are limited to True/False/Zero conditions only. The condition is always checked on the stack top element (naturally, conditional commands that use the stack pop once upon execution).

More advanced conditional expressions are supported by the pre-compiler environment. Refer to sections 13.4.3 and 13.4.5 (Part III) for further information.

Program flow control commands may only be executed from within a macro code (i.e. these commands are not supported from communication terminal).

Example:

- *The following low-level macro code segment demonstrates a subroutine that waits for end of motion condition in the X motor of the controller. The subroutine returns if the X motor is not in motion. While in motion the subroutine is in an infinite loop.*

  Calling for this subroutine from another code segment is also shown.

```
' ********************************************************
' ** Called Subroutine: WEM__X - Wait for end of motion on X motor.
'
#WEM__X:
'
' Wait for No motion in X motor:
' 1) Push X motion status (XMS parameter).
' 2) Push constant '1' (motion status bit 1 means motion ON or OFF).
' 3) Execute '&' operator to extract bit #1 and store the bit on the stack top.
' 4) Execute a JumpTrue command, to the label 'WEM_X'. i.e., if XMS
'    != 0 (condition satisfied), the program will jump back to the
'    sub start location. The program will continue when the jump condition
'    is false (i.e. XMS  = 0, meaning the motor is not in motion).
' ----------------------------------------------------------------------------
XMS};0};!=;XJT,#WEM__X
'
' End condition met, so return to calling subroutine.
' ------------------------------------------------------------
```

```
XRT
‘
 ‘ ******************************************************
‘ ** Calling Subroutine: MOVE_X – Start motion on X motor and wait
‘ ** for end of motion, by calling the WEM_X subroutine.
‘
#MOVE_X:
‘
‘ Initiate motion on X motor:
‘ ---------------------------------
XMM=0;XSM=0;XSP=20000;XAC=100000;XMO=1;XBG
‘
‘ Wait for end of motion on X motor using the Call Subroutine function.
‘ Note that the ‘CS’ command takes a label as a parameter. The parameter
‘ is separated by a single comma. Note that when a label is used as a
‘ parameter no ‘:’ is used (the’:’ is used only for label definition).
‘ ------------------------------------------------------------------------------
XCS, #WEM__X
‘
‘ End condition met, so return to calling subroutine.
‘ ------------------------------------------------------
XRT
‘
‘ ******************************************************
```

Important Note:

- *All program flow control commands described in this section are naturally executed from within a macro code (X and Y macros). It is clear that when a JP command (for example) is executed from an X macro, the jump is relevant for the X macro pointer only, while when the same command executed from within a Y macro will affect only the Y macro pointer, and so on, for all macros. This logic is of course valid for all the above mentioned flow control commands (JP, JT, JF, JZ, CS, CT, CF and RT).*

  Due to this, the preceding 'X' character before the command itself (e.g.'XRT' shown on the example above), has no actual meaning, and the same result will be also if the 'Y' or other legal axis prefix characters where used (including group prefixes). The preceding axis indicating character ('X' or 'Y') is still needed for the internal interpreter logic, but have no other functional meaning.
- *As a rule, try to stick to these strict and clear logic definitions when selecting the preceding character. For example, if a function is related to only the X macro, use 'X' as a preceding character. If a function is related to only the Y macro, use 'Y' as a preceding character.*

10.8 Wait and Internal State Inquiry Functions

In normal programming sequences, it is often required to wait for some events or special conditions to happen. There are three ways of programming a wait sequence in the low-level macro.

- Using simple (standard) commands to inquire about the required state, pushing it to the stack, and then perform some conditional statement (or high level 'if' blocks).
- Using a special state inquiry command QG (automatically inquire the relevant state according to the commands parameter, including extraction of relevant bits from a byte word or long data, and pushing it to the stack), and then perform some conditional statements (or high level 'if' blocks).
- Using a special wait function, QW, that automatically enters to an internal wait condition, until the desired condition is satisfied.

The tradeoff between each method is implementation simplicity, against flow control. For example, the QW command is very simple and short in writing, but does not include time out test (i.e. there is no way to exit from the command before the condition is met). Another case is if more then one condition is required to be tested or waited for. In this case a simple loop should be used.

Both the QW and QG commands share the same parameters (internal state conditions). Table 37 describes the Wait and State inquiry commands supported by the FlexDC low level macro engine.

Keyword	Description
QW	Waits till a specified internal state will be set (or cleared).
QG	Gets the value of a specified internal state (variable). The desired state is provided as a parameter or as a stack argument.

Table 37: Macro Program Wait and State Inquiry Keywords

It should be noted that all the state variables are actually bits of existing keywords (such as “Output Bit 1” which is OP(0)).

The QG command returns the state value as FALSE (0) or TRUE (1).

The QW command holds the macro execution until the state is satisfied.

The following table presents the currently available list of internal states (variables). The Keyword column are the keywords supported by the QW or QG commands. The State Mnemonics Code column is the code for each state value, when using the ‘$define’ directive, supported by the pre-compiler environment (see chapter 13, Part III). The condition that is referred can be either the extraction of one of the bits, or to the value of the actual parameter. The condition can, of course, be the not condition, see Table 38.

Keyword	State Mnemonic Code
MS	0
SR	1
IP	2
OP	3
MO	4
MF	5
QR	6
TD	7
EM	8
EC	9
QC	10

Table 38: Macro Program, Internal Wait Conditions

The parameter sent to the QW or QG command, is actually a formula made of the following:

Parameter = (Axis * 100000) + (Code * 1000) + (Bit * 10) + Logic

Axis:

1➔2. (X,Y – respectively).

Code: 0➔10. See Table 38 for relevant code.

Bit: 0➔32. Bits 1 until 32 are the respective bits of the keyword. If '0' was chosen as bit, it states we are waiting for the keyword to either '==0' or '!=0', according to the Logic.

Logic: 0➔1. Wait for active high, or active low. (or != 0, ==0 when bit = 0).

Example:

- *If it is required to wait until output 3 is active high, send the following:*

XQW,103030

Explanation:

Axis = 1 - therefore we are dealing with axis 1

Code = 3, therefore we are dealing with 'OP'

Bit = 3, therefore we are waiting for bit 3 on 'OP' to turn active high

Example:

- *If it is required to wait until the 3rd output went active low, send XQW,103031.*

The state mnemonics can be used instead of the state number only from within a macro (not in communication clauses). They are converted by the pre-compiler to the related standard constant (numbers) before downloaded to the controller.

Mnemonics are not allowed as a communication clause. The FlexDC will fail to interpret them since it only gets standard constants as state numbers. Refer to chapter 13 for defined constants.

Note:

- *For non-axis related keywords (i.e.OP) the axis sent is irrelevant (can be anything between 1-10), but on the other hand MUST be sent.*

Examples:

- *The following commands wait for digital input #1 to be ON (TRUE) and OFF (FALSE) in the FlexDC:*

```
XQW,102010      ‘ Wait for input #1 ON.
XQW,102011      ‘ Wait for input #1 OFF
```

10.9 Timer Functions

The FlexDC has independent timers, each being updated (decremented) by one, on every hardware sample time. The sample time of the FlexDC is approx 122[μs].

These axis related timers (one per axis 32 bit, positive only) are updated once the dedicated TimerDown (iTD) keyword is called. The axis-related TimerDown (iTD) parameter will Wrap, if the iTD is bigger than 72 hours.

Note:

- *The timers are not updated by the real time kernel of the FlexDC, but by the call to the enquiry of iTD. They do not require a macro to be running. However, the values of timers may be changed to any valid value (32 bits) from both communication and macro programs.*

Table 39 describes the timer functions supported by the FlexDC.

Keyword	Description
Tdi	Timer down axis related variable. Consists of 32 bits, positive only (i.e. 0 ÷ + 2147000000).

Table 39: Macro Program Timer Keywords

Example:

- *The following commands is a simple example for implementing a 8192 sample time delay (1 second), using XTD, and the* ***TimerZero*** *state condition:*

```
XTD=8192        'Set time for 1 sec
XQW,107000      'Wait for timer to be 0
```

First, the XTD is initialized to 8192, then the QW function is called, waiting for timer #1 to be zero.

11 FlexDC Low-Level Macro Program

11.1 Macro and Motions

In the FlexDC the macro execution and the axis motions are completely independent.

If a motion fails for some reason (usually, the motor is disabled in this case) it does not directly affect the macro execution.

Note that if after the motor was disabled a begin motion command is issued, the macro will stop and report a run-time-error, because a BG command is not valid when motor is disabled. But again, this is not necessarily because of the motor failure.

However, since this linkage is sometimes required, the FlexDC supports a dedicated feature: a dedicated keyword (QK) that halts the macro execution and all motions of ALL axes. Upon receiving a BQK command, the controller immediately stops all motions and macro programs.

11.2 Macro Syntax Check and Run-Time-Error

The FlexDC does not perform any macro syntax check when the macro is downloaded or before it is executed. Only a limited syntax check (compilation) is performed by the Nanomotion Shell Application before the macro is downloaded.

However, during macro execution (at run-time), each executed clause is checked as part of its interpreting process. In case of an interpretation error, the controller performs the following:

- Assigns the Error Code to QC. This keyword (axis related) identifies the last macro run-time-error code (the same as the EC codes).
- Stops the macro execution (as with the QH keyword), with the macro pointer (QP) pointing to the clause with the run-time-error. This feature is very important for debugging a macro program.

11.3 Macro Size and Number of Labels

The FlexDC macro is saved on the hardware Flash Memory as a linear buffer. This macro buffer is 16KB long. There is no direct limitation on the number of labels that are supported by the FlexDC macro program. The only limitation is the total number of macro bytes. Note that each label definition consumes 19 bytes from the macro buffer.

11.4 Macro Download Format

The format of the macro download protocol is propriety of Nanomotion Ltd., and can be supplied upon request.

However, the High-Level SCServerInterface DCOM communication interface supports functions for downloading High Level macro programs (compilation + download) and for the downloading of Low Level macro programs (download only). Refer to "Part V– SCServer COM/DCOM Interface Library" for further information.

12 Integrated Development Environment

12.1 General

With the Nanomotion Shell Application the user can simply access all the controller commands, and perform all the operations that the FlexDC supports. Refer to the "Quick Start" chapter in the "FlexDC User Manual" for a brief description of the Nanomotion Shell Application program or refer to "Part IV– Nanomotion Shell Application" for a detailed information.

Part IV thoroughly covers all the details of the Nanomotion Shell Application (see Figure 13), related to macro programming support for the development, downloading, and debugging of FlexDC macros.

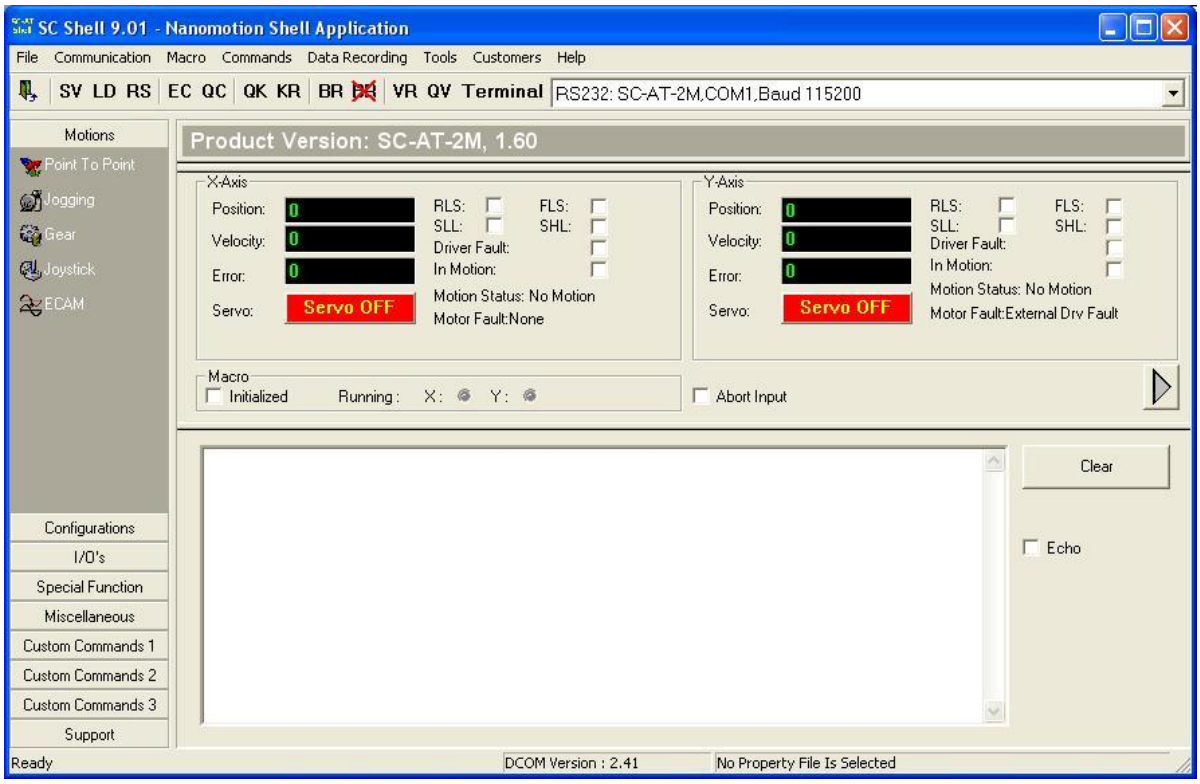


Figure 13: Nanomotion Shell Application Main Screen

12.2 Writing and Editing FlexDC Macro Files

The Nanomotion Shell Application supports an integrated "Srcedit" – the Macro File Editor Application for writing and editing and debugging FlexDC macro programs (see Figure 14). This Macro File Editor Application is installed together with the Nanomotion Shell Application.

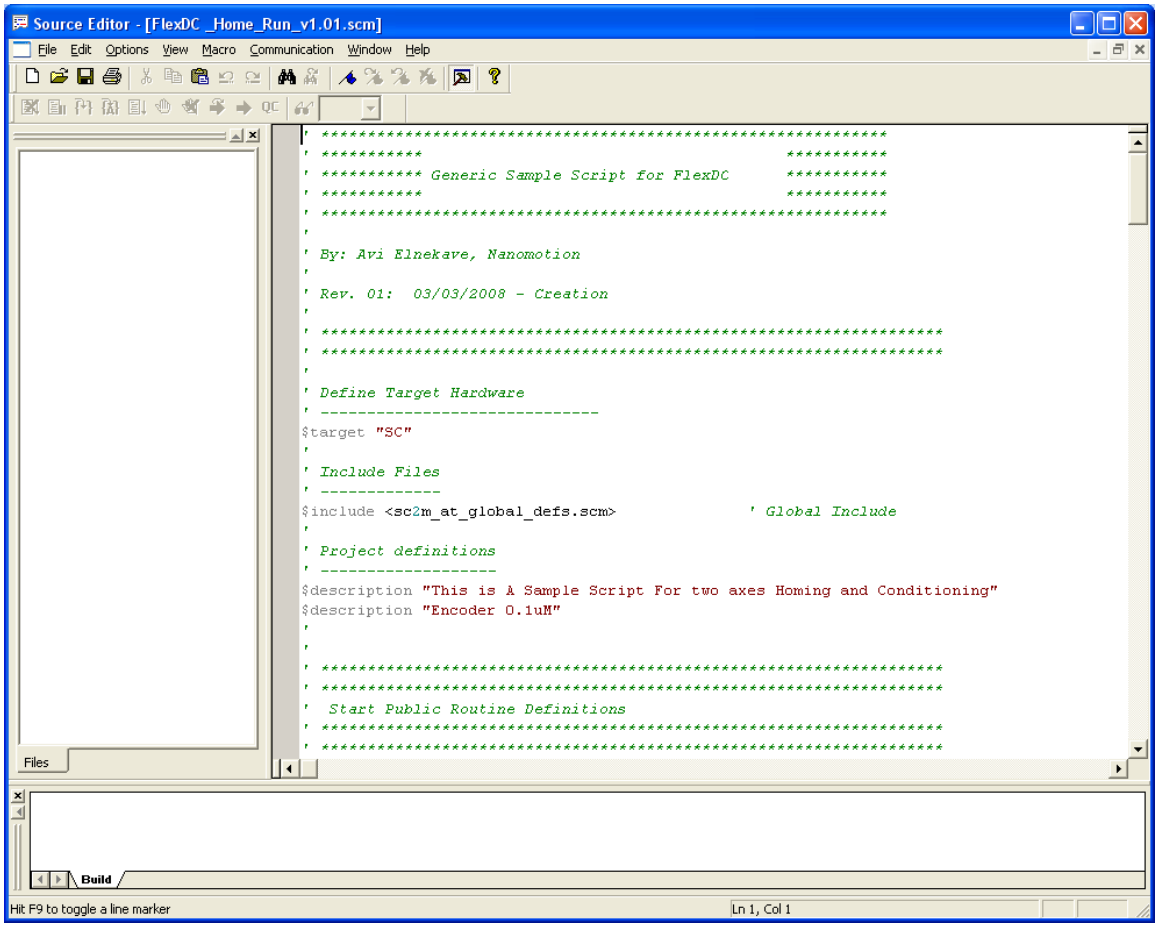


Figure 14: "Srcedit" – the FlexDC Macro File Editor

12.3 Shell Support for Downloading Macro Files to the FlexDC Hardware

The Nanomotion Shell Application supports a user-friendly interface to allow downloading new macro files to the FlexDC hardware.

12.3.1 Download a New Macro

For downloading a new macro to the FlexDC hardware, perform the following steps:

1. Verify that the FlexDC is connected to a communication line, and the communication link is active, refer to Figure 13 (also, refer to the "FlexDC User Manual" for "Quick Start" instructions).
2. On "Macro" menu (on the Nanomotion Shell Application main screen) click "Download Macro".
3. An "Open File" dialog box appears, letting the user to select a desired macro file (default macro files extension is ".SCM").
4. Select the desired macro file and press open. The Nanomotion Shell Application automatically opens the selected file, pre-compiles the program, downloads the macro buffer to the FlexDC hardware memory, and initializes the FlexDC program.
5. A "Download completed successfully" message appears, if the macro download prosess is successful. The program (macro) is now ready for running. Note that after loading a new program, the AUTOEXEC is not started automatically, but only after power up.
6. In order to save the downloaded program to the FlexDC Flash Memory, use the XSV command via the Nanomotion Shell Application terminal window). **A program that is not saved to the Flash Memory will be lost after power up**.
7. If it is required to start an automatic execution of the macro program, switch the FlexDC power OFF and ON again to restart its real-time software. Note that you can still run any valid subroutine name by using the debugger or by issuing an XQE,#<LABEL> command via the terminal window. After downloading a program, issuing an XQE command (with no parameters) starts program execution from the first macro line.

12.3.2 Download a New .DAT File

A ".DAT" file is the file generated by the pre-compiler. To download a new ".DAT" file, perform the following steps:

1. Verify that the FlexDC is connected to a communication line, and the communication link is active, refer to Figure 13 (also, refer to the "FlexDC User Manual" for "Quick Start" instructions).
2. On "Macro" menu click "Download DAT Macro".
3. An "Open File" dialog box appears, letting the user to select a desired macro file (default macro files extension is ".DAT").
4. Select the desired macro file and press open. The Nanomotion Shell Application automatically opens the selected file, pre-compiles the program, downloads the macro buffer to the FlexDC hardware memory, and initializes the FlexDC program.
5. A "Download completed successfully" message appears, if the macro download process is successful. The program (macro) is now ready for running. Note that after loading a new program, the AUTOEXEC is not started automatically, but only after power up.
6. In order to save the downloaded program to the FlexDC Flash Memory, use the XSV command via the Nanomotion Shell Application terminal window). **A program that is not saved to the Flash Memory will be lost after power up.**
7. If it is required to start automatic execution of the macro program, switch the power OFF and ON again to restart its real-time software. Note that you can still run any valid subroutine name by using the debugger or by issuing an XQE,#<LABEL> command from the terminal window. After down loading a program, issuing an XQE command (with no parameters) starts program execution from the first macro line.

Notes:

- *The pre-compiler searches for several types of errors in the downloaded program. If an error is found, an error message is issued, and a window is opened describing the error reason and source (including line number that caused the error). An error file is also generated describing the errors found during the pre-compile process. The error file name will have the same name as the program name, with an '.ERR' extension (see full description of pre-compiler process later on in this chapter).*
- *The Nanomotion Shell Application supports pre-compiling of macro program files even without communication to a target FlexDC hardware. This may be useful to allow writing and initial syntax testing when the hardware is not available. On Macro menu (on the Nanomotion Shell Application main screen), click "Pre-compile Macro". The Nanomotion Shell Application actually performs the normal "Download Macro" procedure, only without downloading the macro buffer to the hardware.*
- *The user can select a default directory in which the Macro Download dialog box opens. On "File" menu, click "File Locations" to define the default file locations. The default file name is saved with the Nanomotion Shell Application setup. Figure 15 shows the "File Locations" definition dialog box. Use the browse […] buttons to open a Windows browser tree dialog box.*

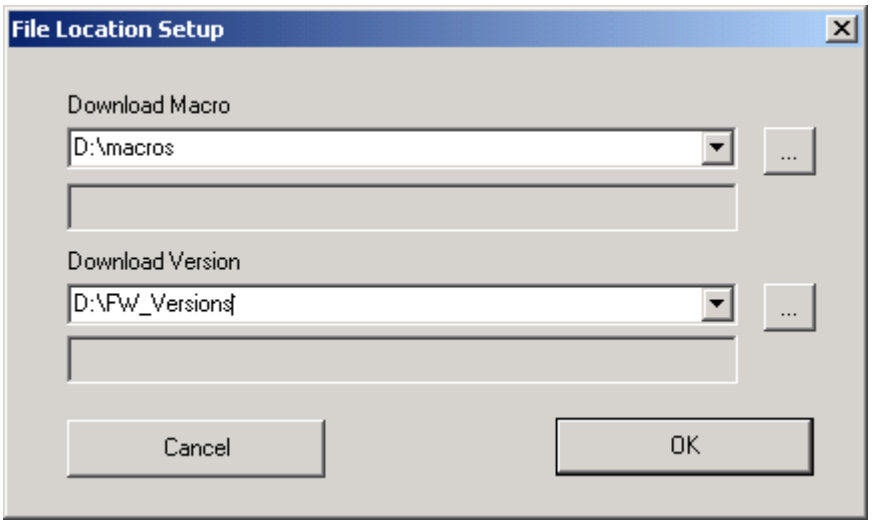


Figure 15: FlexDC Shell File Locations Setup Dialog

12.4 Srcedit Macro Debugger Environment Features

12.4.1 General

The Srcedit is a powerful debugging environment, and the FlexDC macro programs are part of this environment. The Srcedit uses the low-level debug features of the FlexDC, while providing an advanced GUI support. Basically, the debugging environment allows the user the following options:

- Opens a macro source file for debugging, with an advanced, color syntax, highlighted source view (both high level and low-level commands are shown in a clear manner).
- Starts and stops program execution, for all axis macros. Fast, single selection combo-box switching, between the macro debugging. Note that all the features described in this list are available for each macro separately.
- Restarts (resets) a loaded program.
- Brakes program execution at any point.
- Traces program execution line by line.
- Animates program execution (auto-trace).
- Uses up to 20 breakpoints (20 for each macro).
- Removes all breakpoints.
- Sets a program pointer to a specific location (line number or label).
- Shows the next executable line ("go to" current pointer location).
- Shows run-time-errors.
- Full access to all controller parameters (read and write) while debugger is active (with the watch frame).
- All the above options can be accessed in several ways: using the debugger window menu, or using the debugger window toolbar, or using the mouse right-click option to open a pop-up menu, or using dedicated accelerator keys.
- It is possible to debug one macro while other macros are running, or to debug more than one macro simultaneously, to test synchronization issues.

Note:

- *Debugging, pre-compiling and down-loading macros with the Srcedit Macro Debugger Environment are available ONLY while the Nanomotion Shell Application is open and communicating with the controller.*

12.4.2 Srcedit Macro Debugger Window

To debug a macro program, run the Srcedit application and follow the next steps:

12. On "File" menu (on the Srcedit main screen, click "Open".
13. An "Open File" dialog box appears.
14. Select macro file to debug (file extension "SCM"). There are two options to debug the opened program:

- If the program is currently in the controller, on "Macro" menu (on the Srcedit main screen), click "Debug Macro".
- If the program is currently not in the controller, first download the macro. On "Macro" menu, click "Save and Download Current Macro", and then click "Debug Current Macro".

In either case, the Srcedit application checks, with the help of the Nanomotion Shell Application, whether the opened macro file matches the current FlexDC program. An error message appears if the files are mismatched. If the test is OK, the main debugger window opens.

Figure 16 shows the Srcedit application debugger window (sample code source view is for demonstration purposes only):

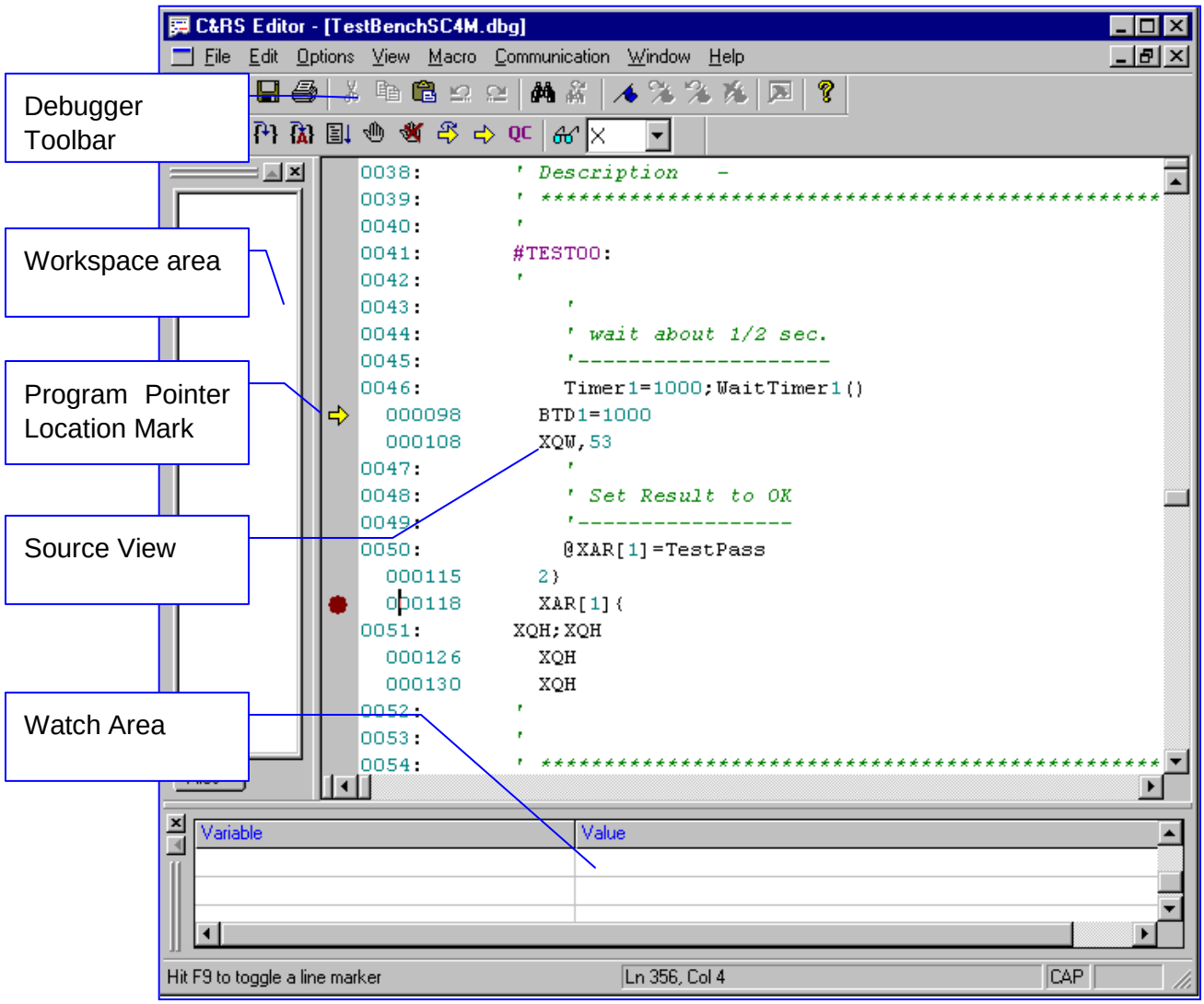


Figure 16: Srcedit Macro Debugger Window

The debugger window has some fields that allow the activation of all the debugger features, and allow user interface. In the following sections, each field is shortly described.

12.4.2.1 Debugger Window – Source View Area

The Source View Area is a read only view that shows the macro source code file, combined with the actual low-level code generated automatically by the pre-compiler.

Original source code lines are shown in dark black color, accept for special keywords that are colored in blue (if, else etc.), labels that are colored in red, and comments that are colored in light green.

Preceding each original source line is the original source code line number, e.g. ‘0041’.

Each original source line is processed by the pre-compiler, and converted to an executable line (note that in some cases, the executable line is identical to a source line). Executable lines are always colored in light gray, to be distinguished from original source code lines.

12.4.2.2 Debugger Window – Source Icons Area

The Source Icons Area is a gray narrow column, located on the left of the Source View window, which is used to mark program pointer locations and breakpoints.

The yellow arrow indicates the current program pointer location. It is always pointing to executable lines (not original source code lines). Note that the pointer location icon is visible only when the relevant macro programs are not running.

The red dot is used to mark breakpoints. Breakpoints are always located next to executable lines (not original source code lines). If the user wants to set a breakpoint on a non-executable line, the Srcedit issues a warning message, and may locate the breakpoint on the next valid executable line if requested.

12.4.2.3 Debugger Window – Toolbar Menu and Pop-Up Menu

The debugger window toolbar menu and pop-up menu (by right-clicking) include 12 different options to operate all the debugger features.

In this section a description of each item is given. Note that the description is identical for all three interfaces mentioned above.

Figure 17 shows the debugger window toolbar:

Figure 17: Srcedit Macro Debugger Window Toolbar

Table 40 describes each toolbar icon (from left to right) as appear in the toolbar image, and the relevant related menu:

#	Description	Related Menu and accelerators
1	Stop debugging, and reset macro program (affecting macros). It is equivalent to the following sequence of commands: XQK;XQI;	Reset Program - [Ctrl+R]
2	Break macro execution (on selected macro only). It is equivalent to the XQH, YQH etc…commands.	Breaks macro program execution. [Ctrl+B]
3	Runs the selected macro line by line (single clause at a time). It is equivalent to the XQTetc...	Trace one step - [F10]
4	Animates the macro program execution. Animation rate is ~ 10 clauses/second. It is equivalent to the XQT etc… commands, 10/sec.	Animate macro execution - [Ctrl+A]
5	Runs the program from the current pointer location. It is equivalent to the XQE, YQE… commands (with no parameter).	Go from current pointer location. [F5]
6	Toggles a breakpoint ON/OFF. Location is selected by the mouse location. The nearest valid executable line is affected. Note that breakpoints are valid for normal run, and also for animate run. Note also that when running normal, when reaching a breakpoint, the program pointer will point to the next executable line after the breakpoint. When animating, the program pointer will point on the breakpoint line (before it was executed).	Insert/Remove breakpoint. [F9]
7	Remove all breakpoints from the controller and from the debugger window.	Remove all breakpoints.
8	Locate the program pointer at the next valid program, clause. Location is selected by the mouse location.	Set next statement - [Ctrl+N]
9	Shows the current pointer location (located at XQP YQP,…accordingly).	Show next statement - [Ctrl+V]
10	Opens a dialog shortly describing possible reasons for the last macro run-time-error.	Show run-time-error [Ctrl+Q]
11	Update Watch Window.	Macro/Update watch list
12	Axis Combo box – selection for the current macro you wish to debug.	N/A

Table 40: Srcedit Macro Debugger Window Toolbar and Menu Functions

12.4.3 Srcedit File Menu

This section is intended to help the user, start editing/creating a new macro (script), and how to work with workspace fetchers.

12.4.3.1 Creating New Macro

All macros must be located in the same folder. To create a new macro, follow the next steps:

1. On the "File" menu click "New". A new "Untitled" file is created, with no extension.
2. On the "File" menu click "Save As", type a file name, select file's location and save the file under the ".SCM" extension.
3. In the created script file add the macro target: $target "SC" (see section 13.3.1, Part III).
4. Add the $include <sc2m_at_global_defs.scm> (see section 13.3.4, Part III).
5. Add the "#AUTOEX" label at the beginning of the script file. Note: At power on, the program begins to run from the first line, regardless of whether this label is added or not, until it reaches the XQH command – which halts the program:

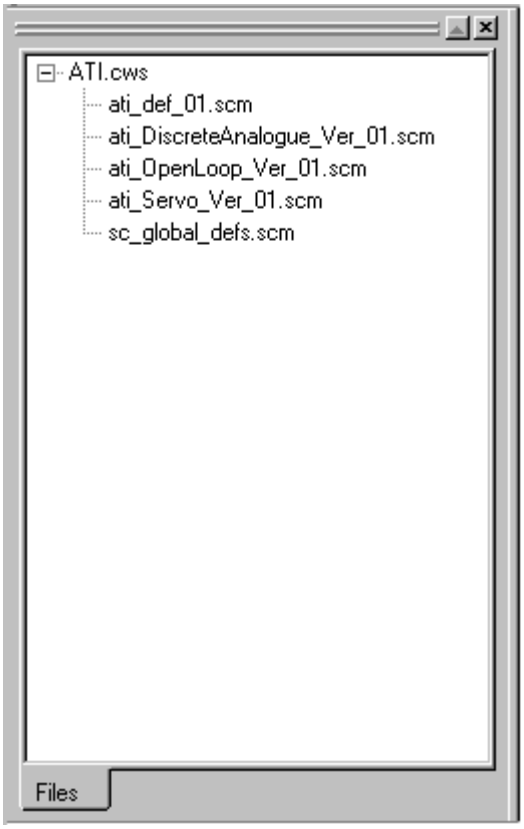
- If the user is interested in running the whole script, the XQH command must be placed at the end of the script.
- If the user is not interested in running the whole script, the XQH command must be placed right after the "#AUTOEX".

6. Write the body of the script file.
7. Place label after the XQH command, in order to resume running the script, if needed. For example, #HOME_X.
8. Save the script. See script example in chapter 14, Part III.

12.4.3.2 Working with Workspaces

The purpose of workspace is to help you manage macros that include more than one file.

For example: If your macro includes two files for definitions and one file of actual code, the best way to manage such a macro is with a workspace.



12.4.3.3 Creating New Workspace

To create a new workspace, on "File" menu click "Workspace", click "New Workspace". This option enables the user to choose or create a new file for the workspace; the file is to be saved with the ".CWS" extension.

12.4.3.4 Open/Close/Save Existing Workspace

- To open an existing workspace: on "File" menu click "Workspace", click "Open" in the "Open Workspace File" dialog box, and choose the workspace to open.
- To save the current opened workspace: on "File" menu click "Workspace", click "Save".
- To save the current opened workspace: on "File" menu click "Workspace", click "Save As" and choose the new name to save the workspace to, with the ".CWS" extension.
- To close the current workspace (without saving): on "File" menu click "Workspace", click "Close". This prompts the user to save the workspace in the following manner: choosing "Yes" saves the workspace, otherwise the changes made in the workspace are discarded.

12.4.3.5 Add/Remove Files to/from Workspace

Removing a file from workspace:

- Select a file in the workspace area. "File" menu click "Workspace", click "Remove File", this removes the file from the workspace.
- Select a file in the workspace area to remove, right click the mouse, select "Remove File" from the menu.

Adding a file to workspace can be done in two ways:

- Drag the file into the workspace area, this will add the file to the current workspace. OR
- On "File" menu click "Workspace", click "Add File" and select the file from the dialog box.

Note:

- *The changes made in the workspace are saved only after the user saves the workspace. Closing the workspace without saving discards the changes.*

12.4.3.6 Open File in Workspace

- Double click the file from the workspace area to open the file for editing.

12.4.3.7 Compiling Workspace

- To compile a workspace the main macro file MUST be open and active (on top of all other open files).
- If a workspace contains more than one macro, then select the macro to compile and opens it. The macro window MUST be open and active (on top of all other open files).

13 The IDE Pre-Compiler Support

13.1 General

The Nanomotion Shell Application includes a built-in Integrated Development Environment (IDE) pre-compiler module. The main purpose of the pre-compiler is to extend the basic features of the low-level language syntax (mainly parsing capabilities), to a more easy to use, high level syntax.

The Nanomotion Shell pre-compiler supports the following features:

- Strip comments, blanks, tabs, and any other non-executable code.
- Target hardware type definition.
- Using the ‘define’ directive.
- Using the ‘include’ directive.
- Using the ‘description’ directive.
- Advanced parsing of mathematical expressions.
- If blocks.
- While loops.
- For loops

The Nanomotion Shell Application automatically activates the pre-compiler, each time a new macro file is being downloaded.

The source file is first scanned to replace all "define"s and combine all the "include" files, then it is stripped from all the comments, spaces tabs and other non-executable code, then the mathematical expressions and special "if", "while" and "for" statements are being processed. Finally, all the labels are searched and replaced by absolute program pointers.

During the pre-compile process, the following files are being generated automatically by the Nanomotion Shell Application (all having the same name and path as the original source file, but with different extensions):

- Extension: ***'.stp'***, an intermediate file used by the pre-compiler.
- Extension: ***'.dat'***, the final buffer actually being downloaded to the controller (this file contains low-level commands only – this file can also be downloaded to controller, see section 13.3, Part III.
- Extension: ***'.dbg'***, the debugger symbols file. This file contains all the original code (including all defined symbols and included files), and all low-level commands. The file is organized such that each (executable) source line is followed by its translated executable low-level code. The Srcedit Macro Debugger uses this file for the debugging process.
- Extension: ***'.err'***, an error file describing errors that were found during the pre-compile process (if there were any).

Since a file being downloaded to the controller is always pre-compiled (no comments, spaces etc.) the actual data buffer, though being ASCII based, is difficult to understand and track. For this reason, in order to debug a FlexDC program, the user should have the original source macro code, and its related ***'.dbg'*** file (being automatically generated by the Nanomotion Shell Application).

In any case, the user can upload the actual macro buffer from the controller by clicking "Upload Macro" on the "Macro" menu (on the Nanomotion Shell Application). Note that the buffer header includes all the descriptive commands (see below) and also the file name and the date of last version download. This may be used for version control of macro code.

In the following sections, all the above features are thoroughly described. Examples are given on each subject to cover all relevant aspects. The parsing logic (to the low-level syntax) is also described.

13.2 Non Executable Code: Comments, Blanks, etc.

When writing a program source code, in order to achieve a clear readable and maintainable code, it is typically accepted (and sometimes even required) to use comments, spaces, empty lines tabs etc. After a while, the macro buffer tends to fill up with comments and non-executable code, which are eventually reduce the final performance of the program. To solve this problem, the pre-compiler, strips from the source non-executable code sections.

The following non-executable codes are removed:

- **Comments:** are defined by the Nanomotion Shell environment as any text that appears in a line following the " ‘ " character sign. A comment
  " ‘ " may appear anywhere in a code file. This includes full lines of comments, or executable code line, followed by the comment sign (on the same line) to describe a specific statement. An example for comment lines is given below:

```
' A full comment line, followed by another one, and an empty line.
' ------------------------------------------------------------------------

    XZI1=10         ' This is a comment within an executable line
    XZI2=11         ' This is also a comment within an executable line
```

- **Empty Lines:** are lines with no text. Empty lines are completely removed by the pre-compiler before downloaded to the FlexDC macro buffer.
- **Blanks and Tabs:** in normal executable statements, the pre-compiler removes blanks and tabs only before the start and after the end of a statement line. Blanks and tabs within a normal executable statements are not removed. In advanced mathematical expressions (lines starting with ‘@’, blanks and tabs are removed also from within the statement itself (see section 13.4, Part III, for further information about advanced expressions parsing).

13.3 Directive Commands

Pre-compiler directives, such as **$define** and **$include**, are typically used to make source programs easy to change and easy to compile in different execution environments. Directives in the source file tell the pre-compiler to perform specific actions. For example, the pre-compiler can replace tokens in the text and insert the contents of other files into the source file. Pre-compiler lines are recognized and carried out before any other action is taken.

The "**$**" sign, defines a directive command. It must be the first nonwhite-space character on the line containing the directive. No white-space characters should appear between the $ sign and the first letter of the directive. Directives include arguments. The comment delimiter " **‘** "must precede any text that follows a directive command (except for arguments or values that are part of the directive).

Note that some directive commands may appear anywhere in a source file, while some are valid only at the beginning of a file.

The **$target** directive must appear only at the beginning of a source file. The **$define, $description** and **$include** directives may appear anywhere in a source file. Like ‘C’ programming, the **$define** directive command applies only to code appearing after its definition (this is also true for defines within include files).

As a thumb rule, use all **$define** directive statements at the beginning of any source file. Use separate include files for define statements, and for sub-routines implementation.

Below are described the directive commands supported by the Nanomotion Shell pre-compiler.

13.3.1 The ‘target’ Definition Directive

The **$target** directive defines the current firmware type and version of the hardware to which the macro is being downloaded to. It is used to define features supported by the current version, internal controller macro buffer size, and possibly other parameters.

The syntax for the directive is:

$target “<product-type>,<product-version>,<buffer_offset>,<buffer_size>”

All the characters between the “” are the actual target definition. Comment may be placed on the same line, after the target definition, preceded by the comment (‘) sign character. For example, the following target directive defines a FlexDC product firmware version 1.41, macro buffer offset 0, and macro buffer size 250000 bytes.

$target "SC-2M,141,0,250000"

Note that a **$target** directive must be defined in order to pre-compile and download a macro. Not defining this directive is an error. The **$target** definition directive should appear before the first executable code line in a source file. This also implies to include files. If an include file contains actual executable code, it must appear after the **$target** definition. Include files that contain only **$define** directives may appear before the **$target** definition.

13.3.2 The 'define' Directive

The **$define** directive may be used to give a meaningful name to a constant in a program.

The syntax for the directive is:

$define identifier "token-string"

The **$define** directive substitutes *token-string* for all subsequent occurrences of the *identifier* in the source file, except for cases where the identifier itself appears as a token in another define statement. In this case the identifier appears as it is. The token string must appear within **" "**. Note that the **"** character may appear as a part of the token string itself (more then once), to allow string parameter definitions.

The **$define** directive must appear before using the *identifier*. Note that if an identifier is used before its definition, the pre-compiler will not detect it as an error. In this case the actual *identifier* string itself is used and not the *token-string*.

One or more white-space characters must separate *token-string* from *identifier*. Any characters appearing between the left **"**defining the *token-string* and the *identifier* are ignored. Note that every character appearing within the *token-string* (between the **" "**) are used, including white spaces, characters, etc.

The following example illustrates the usage of the **$define** directive.

Example:

- *Consider the following simple start jog motion sequence. Speed is set to 20,000, motion mode set to 1 (Jog in FlexDC), motor is enabled and commanded to begin the motion:*

```
XSP=20000              ‘ Set Speed
XMM=1;XMO=1;XBG ‘ Set Motion Mode, Enable motor, Begin Motion
```

- *This sequence can be also implemented by using the following definitions:*

```
$define XFastHomeSpeed  "20000"  ' Define Fast Home Speed constant
$define XMMJog  "1"                    ' Defines Jogging Motion Mode
$define XBeginMotion  "XMO=1;XBG"        ' Define XBeginMotion Command
```

- *The pre-compiler translates the following commands:*

```
XSP=XfastHomeSpeed                    ‘ Set Speed
XMM=XMMJog                            ‘ Set Motion Mode
XBeginMotion                          ‘ Enable motor and start motion
```

- *to exactly the same sequence of commands as appears above.*

13.3.3 The ‘description’ Directive

The **$description** directive may be used to include comment lines within a macro file, that are downloaded to the controller buffer and saved with the program on the target hardware memory (unlike normal comments, that are stripped during the pre-compile process, and are not downloaded to the macro buffer).

The syntax for the directive is:

$description “descriptive string”

All the **$description** directive statements in a source file are searched for by the pre-compiler. The *descriptive strings* are then placed in a special place, in the beginning of the macro buffer. The *descriptive string* must appear within **“ ”**. Note that the **“** may be a part of the string.

The **$description** directive may appear anywhere in a source file. Nevertheless, during download, all descriptive strings are located at the head of the macro buffer. Each **$description** statement is presented as a single line in the macro buffer.

Descriptive strings are contained in an uploaded macro buffer (on the "Macro" menu, click the "Up Load Macro"). The user may also inquire all directive strings with the BQV command via the terminal.

Note:

- *Since descriptive strings are downloaded to the controller macro buffer, they consume macro buffer space (unlike normal comments). Use them carefully.*

The following example illustrates the usage of the **$description** directive.

Example:

- *The use of **$description** directive statements to define program version control. The following code fragment demonstrates a simple program header that includes descriptive statements, and hardware target definitions.*

```
'
$description  " Test SC_3M Controller, Rev.01"
'
$target  "SC-2M,140,0, 250000"
'
' Program Code
….
```

Issuing an XQV command in this case, yields the following controller response, see Figure 18:

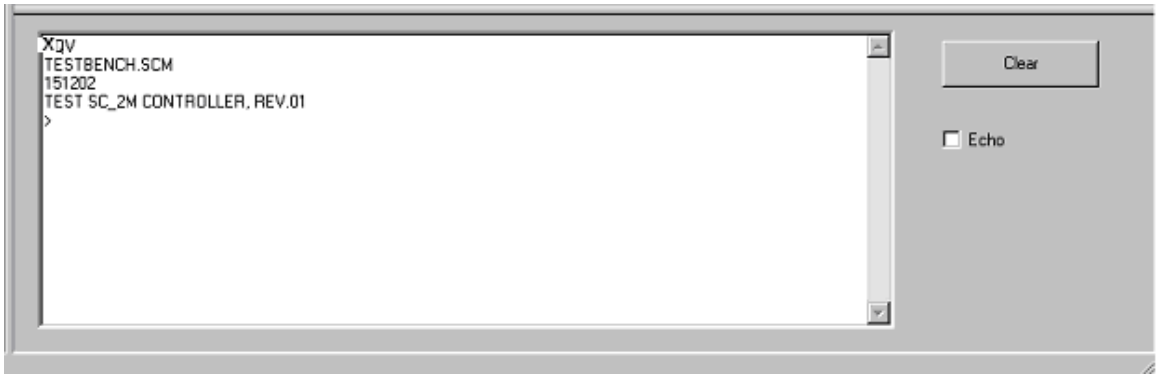


Figure 18: Reporting Descriptive Directive Information

This response includes the following information: The command echo (XQV), followed by the file name used to download the program (TESTBENCH.scm in this case), followed by all descriptive strings (each statement in a single line).

13.3.4 The 'include' Directive

The **$include** directive tells the pre-compiler to treat the contents of a specified file as if those contents had appeared in the source program at the point where the directive appears. You can organize constant definitions and subroutine implementations into include files and then use **$include** directives to add these definitions to any source file. e.g. using global parameter definitions, and for example, homing routines.

The syntax for the directive is:

$include <***path-spec***>** or **$include *"path-spec***"**

The *path-spec* is a filename optionally preceded by a directory specification. The filename must name an existing file. The **$include** directive instructs the pre-compiler to replace the directive by the entire contents of the specified include file. It may appear anywhere in a source file. Note that the include file contents is places at the point where the directive appears. Nested include statements (include statement within an included file) are not allowed.

13.4 Advanced Expressions Parsing

13.4.1 General

As described earlier, the FlexDC is designed with a simple fast real time execution engine, supported by a powerful programming development and debugging environment, the FlexDC.

An important part of the development environment is the pre-compiler described by this chapter. In addition to the directive commands supported by the pre-compiler, another powerful feature is its ability to support advanced expressions parsing (not allowed in the low-level FlexDC language syntax).

The pre-compiler currently supports the following expressions parsing:

- Mathematical expressions.
- If blocks.
- While loops.
- For loops.

Any advanced syntax line should be preceded by the "@" sign. The "@" character must be the first nonwhite-space character on the line containing the expression. No white-space characters should appear between the "@" sign and the first letter of the high level command (if, while, for, etc.). Note that an advanced expression statement can not contain "**;**" (except the "for" loops).

These features are described in the following sections.

13.4.2 Mathematical expressions

Mathematical expressions are statements that include one or more mathematical operators (see list of supported operators below), combined with operands (constants and parameters), to construct a mathematical sentence.

Any mathematical expression statement line should start with the "@" character symbol. The syntax for the mathematical expression is:

@ variable = expression

The *variable* may be any valid FlexDC argument name allowed to be assigned with a value (i.e. parameters and arrays).

Expressions may include operators, constants, parameters (standard and arrays) and commands as operands. When an expression contains an FlexDC command that receives a parameter (separated from the command name by the comma "**,**" char), the command usually pushes the result to the stack by itself.

The following operators are currently supported within expressions:

- Valid binary operators: **+,–,*,/,&,|,^,>,<,==,!=,>=,<=**
- Valid unary operators: **-,||,~,!**
- Brackets: **( )**

Note:

- *Unary operators can be used only on single operands and not on expressions (e.g., not before brackets).*
- *The parser handles mathematical priority as follows:*
  - First mathematical priority is given to unary operators.
  - Second mathematical priority is given to ***** , ***/*** , and **&** over the other binary operators.
  - For other priority use brackets.

Example #1:

- *Initializing array members XPA1 ÷ XPA3 and compute a value for XPA4. The value of XPA4 is divided by 2 and stored in XSP. The value of XSP is then multiplied by 10 and stored in XAC.*

```
XPA1=2000; XPA2=1000; XPA3=4   ‘ Initialization
@XPA4=XPA3*(XPA1*2+XPA2)          ‘ Compute value of XPA4
@XSP=XPA4/2                       ‘ Store for XSP
@XAC=XSP*10                       ‘ Store for XAC
```

Note that rules of FlexDC operator parameters range implies. In the above code for example, the " ***** " operator supports multiplication of 16 bit by 16 bit numbers, and the "***/***" operator supports division of 32 bit by 16 bit numbers (see section 10.4, Part III, for further information).

The actual low-level code generated by the FlexDC compiler environment is shown in Figure 19, for reference. The debugger window shows both the original code lines (in black color), including the comment lines, followed (for each source line) by the low-level compiler implementation.

Note how blank spaces are stripped when evaluating the mathematical expressions.

The user can step line by line (on the low-level code lines) with the debugger, and on each step inquire the stack value (using the BQN – report stack command), for debugging the code.

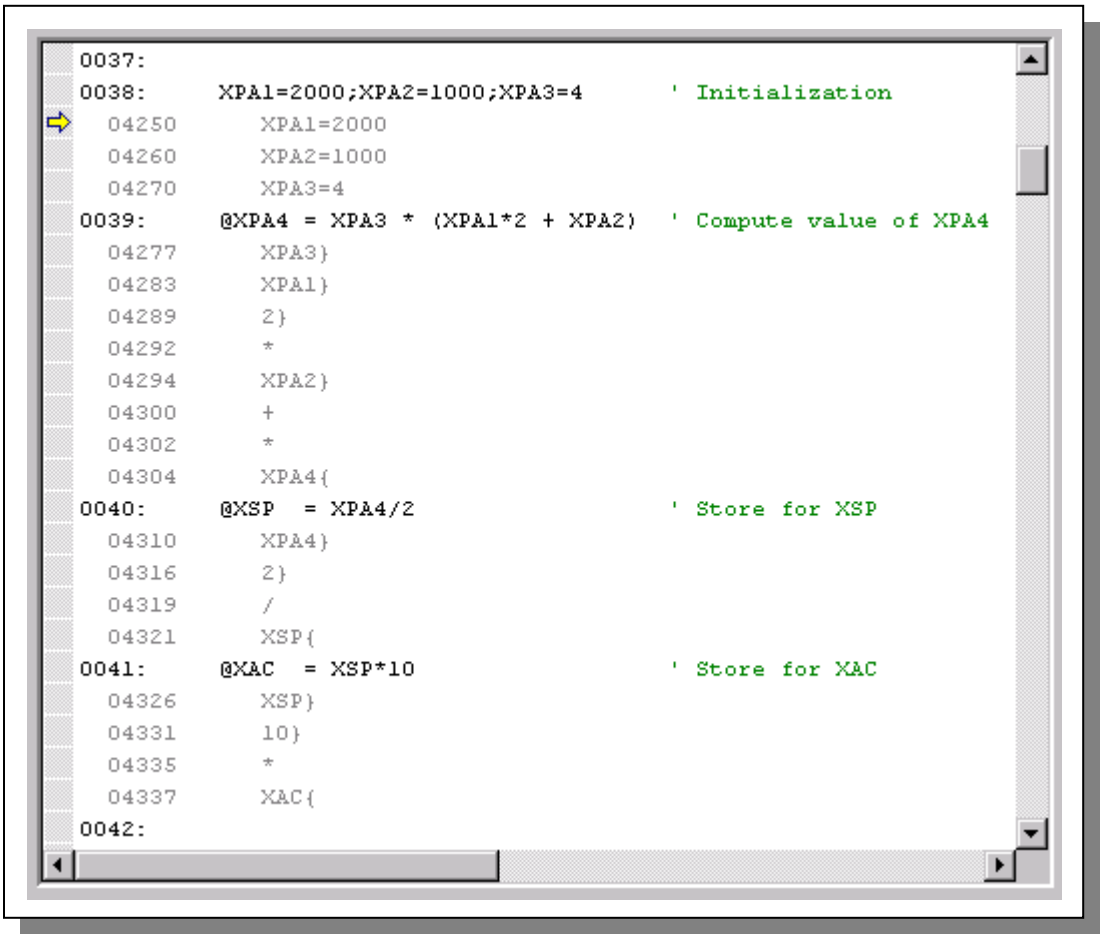


Figure 19: Mathematical Parsing Example

Example #2:

- *Another simple example below shows a combination of the **$define** directive with a mathematical expression to create a user defined variable ('bInput3') holding the Boolean value of digital input port #3.*

$define bInput3 "XPA10"

@bInput3 = XIP & 4

The global Parameters Array (PA[10]) is used to define a user variable ('bInput3'). It is then used as the left side argument in a math statement. The math expression takes the value of XIP (a word bit array containing all digital input ports), extracts bit #3, using the bit-wise & operator with argument '4', and then store the result in bInput3 (actually XPA10). bInput3 can later be used in any other math, simple or conditional expressions.

13.4.2.1 Using the ANS keyword

The pre-compiler supports a special keyword, the **ANS**. This keyword instructs the compiler to use the last value found on the stack top. Note that the **ANS** keyword may appear only at the begging of the right hand side of math expressions. It is useful in cases where a previous statement lest the stack with some result, and to save the pop and push instructions.

The following example demonstrates the usage of this keyword, for the previous example:

Example:

```
$define bInput3 "XPA10"
XIP}
@bInput3 = ANS & 4
```

As before, the global Parameters Array (PA[10]) is used to define a user variable ('bInput3'). XIP is then pushed to the stack top. The math expression recognizes the ANS keyword, indicating that the stack already contains a parameter, so it just extracts from it bit #3, using the bit-wise & operator with argument '4', and then store the result in bInput3 (actually XPA10).

13.4.2.2 Using FlexDC Commands as Operands

A right hand side of a mathematical expression may include a FlexDC command that receives parameters. The following example demonstrates the usage of this syntax. We are using again the previous example, to get a variable holding the Boolean value of digital input #3.

Example:

```
$define bInput3 "XPA10"
@bInput3 = XQG,30
```

*The XQG command (inquiring internal state value) is used here to get the value of digital input #3 (parameter value 30 is **INPUT_3**). The value is pushed to the stack top by the command itself. Is then assigned to bInput3 (actually XPA10).*

13.4.3 If Blocks

If blocks are used for conditional code execution. As with math statement expressions, the **if**, **else** and **endif** statements should start with the "@" character symbol.

The syntax for the **if** block statement is:

@**if (**expression**)**

Statement1…

@**else** [Optional]

Statement2…

@endif

The **if** keyword executes statement1 if the expression is true (nonzero). If **else** is present and the expression is false (zero), it executes statement2. After executing statement1 or statement2, control passes to the next statement.

Nested **if** blocks are supported. It is possible to include **while** and **for** loops within an **if** statement, given that they do not cross the boundaries of the **if** block (and each other's).

Statements may be any valid, normal or math expressions. Valid operators within the **if** statement itself (the conditions) are:

- Valid math operators: **+**,**–**,*****,**/**,**&**,**|**,**^**,**~**,**||**,**!**
- Valid logical operators: **>**,**<**,**==**,**!=**,**>=**,**<=**

Example #1:

- *This example shows a simple usage of an if block to compute an ABS value of a parameter:*

```
XPA1=-100          ‘ Assign a temporary negative value to XPA1
@if (XPA1 < 0)          ‘ Simple (< 0) If expression
@XPA1=-XPA1          ‘ Statement 1
@endif          ‘ End if.
```

The result of this code block will be a positive value of 100 XPA1 (initialization of XPA1=-100 is for the example purpose only).

Example #2:

- *This example shows a simple implementation of a saturation function. The saturation value is given by a parameter:*

```
XPA1=-100          ‘ Assign a temporary negative value to XPA1
XPA10=45          ‘ Saturation value
‘
‘ Check negative saturation
‘ ------------------------------
@if (XPA1 < -XPA10) ‘ Simple (<) If expression
@XPA1=-XPA10       ‘ Statement 1
@endif          ‘ End if.
 ‘ Check positive saturation
‘ ------------------------------
@if (XPA1 > XPA10)  ‘ Simple (>) If expression
@XPA1=XPA10        ‘ Statement 1
@endif          ‘ End if.
```

13.4.4 While Loops

While loops are used to execute repeated block of statements, until some condition expires. As with math statement expressions, the **while**, **continue**, **break** and **endwhile** statements should start with the "@" character symbol.

The syntax for the **while** block statement is:

Example:

@**while (**expression**)**

Statements…

@**continue** [Optional]

@**break** [Optional]

@endwhile

The **while** keyword executes statements repeatedly until the expression becomes 0. The **endwhile** keyword must end any while block. Nested **while** loops are supported. It is possible to include **if** statements and **for** loops within a **while** statement, given that they do not cross the boundaries of the **while** block (and each other's). Several **continue** and **break** commands are allowed within a **while** loop. **Continue** and **break** commands are valid only within **while** and **for** loops. Statements may be any valid, normal or math expressions. Valid operators within the **while** statement itself (the conditions) are:

- Valid math operators: **+**,**–**,*****,**/**,**&**,**|**,**^**,**~**,**||**,**!**
- Valid logical operators: **>**,**<**,**==**,**!=**,**>=**,**<=**

Example #1

- *Showing a simple infinite loop using a **while** statement, counting seconds:*

```
@while (1)                    ‘ Infinite loop
    XTD=8192; XQW,107000      ‘1 Second Delay
    @XPA1=XPA1+1
@endwhile
```

Example #2:

- *Showing a simple **while** loop with internal termination test using an **if** and **break** statements.*

```
XPA1=0
    @while (1)                    ‘ Infinite loop
        XTD=8192; XQW,107000      ‘ 1 Second Delay
        @XPA1=XPA1+1              ‘ Increment seconds counter
        @if (XPA1 > 20)           ; Test for expired time (20 seconds)
            @break
        @endif
@endwhile
```

13.4.5 For Loops

For loops are used to execute repeated block of statements, until some condition expires. As with math statement expressions, the **for**, **continue**, **break** and **endfor** statements should start with the ‘@’ character symbol.

The syntax for the **for** block statement is:

```
@for( init-expr; cond-expr; loop-expr )
    Statements…
    @continue [Optional]
    @break [Optional]
@endfor
```

First, the initialization (*init-expr*) is evaluated. Then, while the conditional expression (*cond-expr*) evaluates to a nonzero value, *statements are* executed and the loop expression (*loop-expr*) is evaluated. When *cond-expr* becomes 0, control passes to the statement following the **for** loop.

The **endfor** keyword must end any **for** block. Nested **for** loops are supported. The (*init-expr*) and (*cond-expr)* expressions must include assignments (i.e. =, see example #1 below). Currently all three expressions (*init-expr*, *cond-expr* and *loop-expr*) are necessary. This means that the 'C' syntax: for (; *cond-expr* ;) is not supported.

It is possible to include **if** statements and **while** loops within a **for** statement, given that they do not cross the boundaries of the **for** block (and each other's). Several **continue** and **break** commands are allowed within **for** loops. The **continue** and **break** commands are valid only within **while** and **for** loops.

Statements may be any valid, normal or math expressions. Valid operators within the **for** statement itself (the conditions) are:

- Valid math operators within assignment expressions:

+,–,*,/,&,|,^,~,||,!,>,<,==,!=,>=,<=

- Valid math operators within conditional expressions:

+,–,*,/,&,|,^,~,||,!

- Valid logic operators within conditional expressions:

>,<,==,!=,>=,<=

See the following examples for using **for** loops.

Example #1:

- *A simple **for** loop, counting time with a delay function.*

```
@for (XPA1=100 ; XPA1 < 120 ; XPA1=XPA1+1)    ' Infinite loop
                XTD=16384; BQW,107000                ' 1 Second Delay
@endfor
```

14 Script Example

14.1 Script Structure

The following script consists of "Homing" and "Full Travel Repetitive Running" for X and Y Axes. The script incluides two files:

- The main macro file (presented here).
- Application specific definition file – definitions for array variables that hold the relevant values of all the motion parameters (PIV, Speeds etc…). This macro presumes the relevant motion data is downloaded by a host PC. The macro sends a message via the RS232 (or CAN, or LAN) to the host PC, regarding how the home procedure ended.

Notes:

- *Sc2m_at_Global_Defs.scm – the Include File must be placed in the same folder with main macro file. It stores definitions of all relevant values of all the motion parameters.*

14.2 Script Content

```
'****************************************************************
' ********** 'Generic Sample Script for FlexDC         **********
' ********** '                                                 **********
'****************************************************************
  '
 'By: Avi Elnekave, Nanomotion
'
 'Rev. 01:  03/03/2008 - Creation
'
'*************************************************************************
'*************************************************************************
   '
 'Define Target Hardware
'---------------------------- '
$target "SC"
'
 'Include Files
'------------ '
$include <sc2m_at_global_defs.scm<          'Global Include
'
 'Project definitions
'------------------- '
$description "This is A Sample Script For two axes Homing and Conditioning"
$description "Encoder 0.1uM"
'
'*********************************************************************
  'Start Public Routine Definitions
'*************************************************************************
'
 **'AUTOEX                                   'Function Called at power on.

 **'Find Home and auto travel from hard Limits
#AUTOEX:
 @XPA[2]=0                                  'Zero home X state
 @YPA[2]=0                                  'Zero home Y state
 'Set Servo Parameters
XCA[7]=222749440                            '2nd order filter parameters damping for 700 Hz ξ=0.6
XCA[8]=86352                                '2nd order filter parameters damping for 700 Hz ξ=0.6
XCA[9]=-34411                               '2nd order filter parameters damping for 700 Hz ξ=0.6
XCA[13]=1                                   'Enable 2nd order filter
XKP=2000
XKI=2500
XKD=30
YCA[7]=222749440                            '2nd order filter parameters damping for 700 Hz ξ=0.6
YCA[8]=86352                                '2nd order filter parameters damping for 700 Hz ξ=0.6
YCA[9]=-34411                               '2nd order filter parameters damping for 700 Hz ξ=0.6
YCA[13]=1                                   'Enable 2nd order filter
YKP=2000
YKI=2500
YKD=30
XMO=0;YMO=0                                  'Set servo OFF for X and Y axes
 'These parameters work for Nanomotion HR motors and AB1A drivers
```

```
 **''Homing subroutine for X Axis
#HOME_X:
'defined velocity [counts/sec];acceleration[counts/sec^2[
'decleration [counts/sec^2 ;[
        XSP=-50000;XAC=10000000;XDC=10000000;
 ' Close the servo loop motor on
        XMO=1
 ' Turn to velocity mode by motion mode MM and special mode SM
        XMM=1;XSM=0
 ' Start jog
        XBG
 ' wait untill dac output>= 15000 (~5 volts (
        @while ( abs(XPO) < 15000(
        @endwhile
 ' In this position we are at the left hard stop,turn off the motor
        XMO=0
 ' wait 100msec
        TimerX=800
        WaitTimerX ()
 ' defined hard stop position -0.5mm(-5000 counts(
        XPS=-5000
 ' Turn motor on wait 10msec and move positive to the right hard limit.
        XMO=1
        TimerX=80
        WaitTimerX()
        XSP=50000
        XBG
 ' wait untill dac output>= 15000 (~5 volts(
        @while ( abs(XPO) < 15000(
        @endwhile
        XMO=0
 ' In this position we are at the right hard stop,turn off the motor
        TimerX=800
        WaitTimerX ()
 ' defined right hard stop + 0.5 mm from the max travel YPA[1{
         @XPA[1]=XPS-5000
 'turn the motor on and switch to PTP mode
        XMO=1
        TimerX=80
        WaitTimerX()
        XMM=0
 'move to home position (0) at velocity of 50mm/sec
   XSP=500000
        XAP=0
        XBG
        WaitForEndOfMotionX ()
        XMO=0
        @if (  XPA[2]==1(
   XJP,#MFT_X
@    endif
 'stop program
   YQH
   XQH
'
```

```
 **'Homing subroutine for Y Axis
#HOME_Y :
'defined velocity [counts/sec];acceleration[counts/sec^2[
'decleration [counts/sec^2 ;[
            YSP=-50000;YAC=10000000;YDC=10000000;
 ' Close the servo loop motor on
            YMO=1
 ' Turn to velocity mode by motion mode MM and special mode SM
            YMM=1;YSM=0
 ' Start jog
            YBG
 ' wait untill dac output>= 15000 (~5 volts (
            @while ( abs(YPO) < 15000(
            @endwhile
 ' In this position we are at the left hard stop,turn off the motor
            YMO=0
 ' wait 100msec
            TimerY=800
            WaitTimerY ()
 ' defined hard stop position -0.5mm(-5000 counts(
            YPS=-5000
 ' Turn motor on wait 10msec and move positive to the right hard limit.
            YMO=1
            TimerY=80
            WaitTimerY()
            YSP=50000
            YBG
 ' wait untill dac output>= 15000 (~5 volts(
            @while ( abs(YPO) < 15000(
            @endwhile
            YMO=0
 ' In this position we are at the right hard stop,turn off the motor
            TimerY=800
            WaitTimerY ()
 ' defined right hard stop + 0.5 mm from the max travel YPA[1{
             @YPA[1]=YPS-5000
 'turn the motor on and switch to PTP mode
            YMO=1
            TimerY=80
            WaitTimerY()
            YMM=0
 'move to home position (0) at velocity of 50mm/sec
   YSP=500000
            YAP=0
            YBG
            WaitForEndOfMotionY()
            YMO=0
            @if (  YPA[2]==1(
    YJP,#MFT_Y
@    endif
 'stop program
   YQH
   XQH
'
```

```
 **'Move X axis full travel back and forth
#MFT_X:
XMO=1
 'set absolute position to full travel YPA[1{
  @ XAP=XPA[1 [
 'start motion
   XBG
            WaitForEndOfMotionX  ()
 ' wait 1 sec
            TimerX=8000
            WaitTimerX ()
 'set absolute position to zero
   XAP=0
 'start motion
   XBG
            WaitForEndOfMotionX ()
            TimerX=8000
            WaitTimerX ()
 'start from begining
            XJP,#MFT_X
            XQH
'
'Move Y axis full travel back and forth
#MFT_Y  :
'
YMO=1
 'set absolute position to full travel YPA[1{
  @ YAP=YPA[1 [
 'start motion
   YBG
            WaitForEndOfMotionY  ()
 ' wait 1 sec
            TimerY=8000
            WaitTimerY ()
 'set absolute position to zero
   YAP=0
 'start motion
   YBG
            WaitForEndOfMotionY ()
            TimerY=8000
            WaitTimerY ()
 'start from begining
            XJP,#MFT_Y
            YQH
 'Halt all
#HALT:
 XMO=0
 YMO=0
 YQH
 XQH
```

15 FlexDC Script Keywords Commands Reference

This chapter presents a complete list of macro related commands supported by the FlexDC, according to tasks, in alphabetical order, including detailed explanations and examples.

15.1 Task Based Reference

This section lists the commands according to their relation to several basic tasks. The list provides a short description of each command.

15.2 Task Description

The commands are grouped to in the following tasks.

- Macro handling keywords.
- Operators.
- Flow control.
- Wait and state inquiry functions.
- Timer functions.
- Automatic routine control functions.
- Remote access over the CAN commands.
- Pre-compiler directive commands and keywords.

15.3 Task Based Command List

Table 41 and Table 42 list all the FlexDC commands according to their task. A given command may appear under more then one task.

15.3.1 Macro Handling Keywords

Keyword	Description
QB[ ]	An array of 20 breakpoints pointers (-1 to disable a pointer and following pointers)
QC	Reports the last macro runtime error (if there was any)
QD	Downloads a macro
QE	Execute macro from the current macro pointer (QP)
QH	Halt macro execution
QI	Initialize macro and its internal variables
QK	Kill macro execution (also stops all motions of both axes)
QL	Loads the macro from the Flash Memory. Automatically after power ON or reset. This command is currently not implemented. Using the LD (for loading parameters) also loads the macro.
QN	Displays the macro stack
QP	Holds the current macro pointer
QQ	Uploads the program stack (queue of return addresses)
QR	Reports the macro status
QS	Saves the macro to the Flash Memory.
QT	Execute single macro clause (trace) from the current macro pointer (QP)
QU	Uploads a macro
QV	Uploads all macro descriptive data and its checksum
QZ	Clears all the numbers stack

Table 41: FlexDC Program Handling Keywords

15.3.2 Operator Keywords

Keyword	Description
QN	Displays the macro stack
QZ	Clears all the numbers stack
}	Push (without argument, duplicates last stack element)
{	Pop (without argument – remove last stack element)
+	Add
-	Subtract
*****	Multiply
/	Divide
||	ABS
+-	Negate
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
!	Logical NOT (result is always 0 or 1)
>0	Is positive
<0	Is negative
=0	Is zero
!0	Is not zero
>	Is greater
<	Is smaller
==	Is equal
!=	Is not equal
>=	Is greater equal
<=	Is smaller equal

Table 42: Macro Program Operators

Note that push and the pop operators must be attached to a parameter name.

15.3.3 Flow Control Keywords

Keyword	Description
CS	Call subroutine at a new macro pointer
CT	Call subroutine if last stack element is TRUE (not zero)
CF	Call subroutine if last stack element is FALSE (zero)
JP	Jump to a new macro pointer
JT	Jump if last stack element is TRUE (not zero)
JF	Jump is last stack element is FALSE (zero)
JZ	Jump to a new macro pointer and clear subroutines stack (to restart the macro with subroutines stack clear)
RT	Return from a subroutine

Table 43: Macro Program Flow Control Keywords

15.3.4 Wait and Internal State Inquiry Functions

Keyword	Description
QW	Waits till a specified internal state will be set (or cleared).
QG	Gets the value of a specified internal state (variable). The desired state is provided as a parameter or as a stack argument.

Table 44: Macro Program Wait And State Inquiry Keywords

15.3.5 Timer Function Keywords

Keyword	Description
iTd	Timers down axis related variable. Consists of 32 bits, positive only. Each element is calculated once the iTD is called.

Table 45: Macro Program Timer Keywords

15.3.6 Remote Access over the CAN commands

Table 46 describes the keywords that allow remote CAN access from within an FlexDC macro program:

Keyword	Description
ZA	Remote Assign parameter. Sends an assignment clause to a remote unit. The parameter to be assigned is the command's parameter. The value to assign is taken from the numeric stack (one item removed).
ZC	Remote Command. Sends a command clause to the remote unit. The command to send is the ZC command's parameter. The command does not affect the numeric stack.
ZI[I]	An array parameter. ZI[1] holds the ID address to which the remote communication addresses (the receive CAN ID address of the remote unit). ZI[2] holds the ID address to which the remote unit will answer (the CAN ID address at which the answer is expected). This array parameter is macro related: XZI[1] is used in macro X, YZI[1] is used in macro Y...
ZM	Sends a string message (limited to 8 characters) to a pre-defined remote, CAN ID address, defined by ZI[1] (ZI[2] is ignored). The message to send is the command's (string) parameter if ' " ' was found , or 1 or 2 (as parameter) for 1 or 2 (respectively) numbers OFF the numbers stack.
ZR	Remote Report parameter. Sends a report clause to a remote unit. The parameter to be reported is the command's parameter. The reported value is pushed to the numeric stack (one item is added to this stack).
ZS	A parameter that holds the status of the last remote unit's response. Can be only reset to zero.

Table 46: Macro Program Remote CAN Access Commands

15.3.7 Pre-compiler Directive Commands and Keywords

Table 47 describes the directives and keywords supported by the pre-compiler:

Keyword	Description
‘	Comment Line.
#	Label definition, Subroutine name.
$define	Global constants define directive.
$description	Macro descriptive comment strings definitions.
$include	Include macro files.
$target	Defines the target hardware to download to.
@for	Defines a "for" loop block statement.
@if	Defines an "if…, else" conditional statement.
@while	Defines a "while" loop block statement.

Table 47: Pre-compiler directive commands and Keywords

15.4 Macro Programming Keywords Reference

This section presents all the controller keywords related to macro programming in alphabetical order, including detailed definitions of each command and examples.

The description of each keyword includes:

- **Purpose:** The operation or task of the keyword.
- **Attributes:** See below.
- **Syntax:** Valid clause syntax.
- **Typical applic.:** Typical use of keyword.
- **Example:** Simple example of the keyword usage.
- **See also:** Related commands.

The following list describes all the valid keyword **Attributes**:

- **Type:** Command / Parameter.
- Axis related[12]: Yes / No.
- **Array * :** Yes (dimension) / No.
- **Assignment *:** Yes / No (read only).
- Receive parameter *: Yes / No.
- **Parameter type *:** Number / String
- **Scope:** Communication / Program / Both
- Restrictions: See below.
- Save to Flash: Yes / No.
- **Default Value:** Yes (value) / No.
- **Range:** Min ÷ Max.

The following list describes all the valid keyword **Restrictions:**

- None.
- No motion.
- Motor is OFF.
- Motor is ON.
- Macro not running (X, Y, …, V).
- ALL Macro's not running.

[12] Axis or Macro related (Keyword's preceding character X,Y, B affects the keyword behavior).
* Applicable for commands only

15.4.1 CS – Call Subroutine

Purpose:
Calls (jumps) to a specified subroutine (given by program pointer or label). After returning from the subroutine (by the RT command), execution continues at the next macro clause.

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Receive parameter:	Yes.
	Parameter type:	Number or Label.
	Scope:	Program.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:
CS,<Label>; CS,<Constant>

Keyword is not axis related, thus XCS, YCS and BCS are equivalent.

Examples:

```
#MAIN:
XCS,#SUB_1      ‘ Calls the SUB_1 Subroutine

#SUB_1:
XMO=1
XRT
```

See also:
RT, CF, CT, #

15.4.2 CF,CT – Call Subroutine If False or True

Purpose:
Call (jump) to a specified subroutine (given by program pointer or label) according to the stack top condition. After returning from the subroutine (by the RT command), execution will continue at the next macro clause. The CF (Call False) command will execute the requested call if the stack top element is false (zero – '0'). The CT (Call True) command will execute the requested call if the stack top element is true (non-zero).

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Receive parameter:	Yes.
	Parameter type:	Number or Label.
	Scope:	Program.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:
CF,<Label>; CF,<Constant>
CT,<Label>; CT,<Constant>
Keyword is not axis related, thus XCF, YCF, etc…are equivalent.

Examples:

```
#MAIN:
XPA1}           ' Pushes the value of XPA1 to the stack top
XCT,#SUB_1      'Calls the SUB_1 Subroutine if XPA1 != 0
YCF,#SUB_2      'Calls the SUB_2 Subroutine if XPA1 == 0
#SUB_1:
XMO=1
BRT
#SUB_2:
YMO=1
BRT
```

See also: RT, CS, #

15.4.3 JP – Jump

Purpose:
Jump to a specified label or pointer location.

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Receive parameter:	Yes.
	Parameter type:	Number or Label.
	Scope:	Program.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:
JP,<Label>; JP,<Constant>

Keyword is not axis related, thus XJP, YJP etc…are equivalent.

Examples:

```
#MAIN:
XJP,#A_1        'Jumps the A_1 label.

#A_1:
XSP=12345
```

See also:
JF, JT, JZ

15.4.4 JF, JT – Jump If False or True

Purpose:

Jump to a specified label or pointer location, according to the stack top condition.

The JF (Jump False) command will execute the requested jump if the stack top element is false (zero – '0'). The JT (Jump True) command will execute the requested jump if the stack top element is true (non-zero).

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Receive parameter:	Yes.
	Parameter type:	Number or Label.
	Scope:	Program.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

JF,<Label>; JF,<Constant>

JT,<Label>; JT,<Constant>

Keyword is not axis related, thus XJF, YJF, etc…are equivalent.

Examples:

```
#MAIN:
XPA1}              ' Pushes the value of XPA1 to the stack top
XJT,#LABEL1 'Calls the LABEL1 Subroutine if XPA1 != 0
YJF,#LABEL2 'Calls the LABEL2 Subroutine if XPA1 == 0
#LABEL1:
XPA2=1
#LABEL2:
XPA2=2
```

See also:

JP, JZ

15.4.5 JZ – Jump Zero

Purpose:
Jump to a specified label or pointer location, and clears subroutines stack (to restart the macro with subroutines stack clear).

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Receive parameter:	Yes.
	Parameter type:	Number or Label.
	Scope:	Program.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:
JZ,<Label>; JZ,<Constant>

Keyword is not axis related, thus XJZ, YJZ and BJZ are equivalent.

Examples:
```
#MAIN:
XPA1=0

XJZ,#MAIN       'Jumps the MAIN label, clears the subroutines stack.
```

See also:
JP, JF, JT

15.4.6 QB – Macro Breakpoint Array

Purpose:

Defines a breakpoint location (pointer) when executing a macro program. Up to 20 breakpoints are supported simultaneously, for ALL macro programs (20 for X , 20 for Y,). Setting a QB element to -1 avoid the check of all the following elements.

QB must be set to the pointer of the first byte of a clause. Otherwise it does not halt the related macro program.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	Yes size: 2x20.
	Assignment:	Yes.
	Receive parameter:	---.
	Parameter type:	---.
	Scope:	Both.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	-1 ÷ Max Macro Pointer.

Syntax:

QB[i]; QBi;

QB[i]= <Number>; QBi= <Number>;

The array index [i] range is (1 ÷ 20).

Examples:

For internal usage ONLY !

See also:

QT, QE

15.4.7 QC – Macro Run-Time-Error

Purpose:
Reports the last macro run-time-error code.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	No (read only).
	Receive parameter:	---.
	Parameter type:	---.
	Scope:	Both.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	---.

Syntax:
QC

Examples:
XQC; YQC;…

See also:
EC

15.4.8 QD – Download Macro Buffer

Purpose:

Downloads the macro program to the internal FlexDC macro buffer. This command is currently active by using the Nanomotion Shell application DCOM communication interface only.

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Receive parameter:	Yes.
	Parameter type:	String.
	Scope:	Communication.
	Restrictions:	No Macro’s are running.
	Save to Flash:	No.
	Default Value:	---.
	Range:	---.

See also:

QU, QV

15.4.9 QE – Execute Macro

Purpose:

Executes a user program from a specified location (program label, or pointer), or from the current location if no parameter is given.

Attributes:	**Type:**	Command.
	Axis related:	Yes.
	Array:	--.
	Assignment:	--.
	Receive parameter:	No.
	Parameter type:	---.
	Scope:	Both.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	--.
	Range:	--.

Syntax:

The QE command can be executed with no parameters, or with a single (Label or constant) parameter:

Syntax	Description
QE	Starts or continues program execution from current pointer location.
QE,<Const>	Start program execution from a given pointer location.
QE,<Label>	Start program execution from a given label.

Examples:

If no parameters are used, the command simply starts (or continues) execution of the relevant macro from the current macro pointer (XQP, YQP,…).

XQE; YQE; …

If a parameter (label or constant) is used, the macro will start (or continue) execution from the specified label or pointer. e.g.

XQE,#XHOME

See also:

QP, QT, QH

15.4.10 QF – Macro Running Status

Purpose:

Reports the macro running status. This is an array report only command.
Currently the following array indexes are reported:

1. Macro Running Index– Mask of running macro's
2. Internally Used.
3. RTE Index of macro has a RTE – Mask of macro's that encountered a Run-Time-Error.
4. Internally Used.
5. Internally Used.

Refer to the QR, for the macro initialization statuses.

Attributes:		
	Type:	Parameter.
	Axis related:	No.
	Array:	Yes.
	Assignment:	No (read only).
	Receive parameter:	---.
	Parameter type:	---.
	Scope:	Both.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

XQF1, YQF2, …etc…

Examples:

```
XQF1      ' Reports the Macro Running Status Register of all axes.
XQF3      ' Reports the Macro RTE Status Register of all axes.
```

See also:

QI,QR

15.4.11 QG – Get Internal State Value

Purpose:

Gets the value of a specified internal state (variable). The desired state is provided as a parameter or as a stack argument. For a list of supported internal states refer to section 10.8.

The QG command returns the state value to the macro number stack as FALSE (0) or TRUE (1) Refer to section 10.8, Part III, for inversing the logic of the returned value.

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Receive parameter:	Yes.
	Parameter type:	Number.
	Scope:	Both.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

QG,<Constant>

Keyword is not axis related, thus XQG and YQG are equivalent.

Examples:

```
XQG,100000  ‘ Reports on the stack if the X axis is in motion.
XQG,200000  ‘ Reports on the stack if the Y axis is in motion.
XQG,200001  ‘ Reports on the stack if the Y axis is not in motion.
```

See also:

QW

15.4.12 QH – Halt Macro

Purpose:
Halts program execution of the relevant macro program.

Attributes:	**Type:**	Command.
	Axis related:	Yes.
	Array:	--.
	Assignment:	--.
	Receive parameter:	No.
	Parameter type:	---.
	Scope:	Both.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	--.
	Range:	--.

Syntax:
QH

Examples:
XQH ‘Halts execution of the X program.
YQH ‘Halts execution of the Y program.

See also:
QP, QT, QE

15.4.13 QI – Initialize Macro

Purpose:
Initialize and the reset macro program status and flags. Note that after initialization the macro is not running. The QI command is called automatically by the DCOM communication interface application each time a new macro program is downloaded to the controller.

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Receive parameter:	No.
	Parameter type:	---.
	Scope:	Communication.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:
QI

Keyword is not axis related, thus XQI, YQI are equivalent.

Examples:
XQI ‘ Resets Macro programs

See also:
QE, QH, QD, QL

15.4.14 QK – Kill Macro and Motions

Purpose:

Halts program execution of the relevant macro program, and stops any motion in ALL motors. The command is equivalent to the two commands: QH; XST; YST ; etc… (refer to "Part II– FlexDC Software and Commands Reference" for further information regarding the ST command).

Attributes:		
	Type:	Command.
	Axis related:	Yes.
	Array:	---.
	Assignment:	---.
	Receive parameter:	No.
	Parameter type:	---.
	Scope:	Both.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

QK

Examples:

XQK ‘Halts execution of the X program, and stops ALL motions.

YQK ‘Halts execution of the Y program, and stops ALL motions.

See also:

QH

15.4.15 QN – Display Macro Stack

Purpose:

Reports the macro program numbers stack. The QN keyword may be used to debug macro execution. The user can inquire the numbers stack form the Nanomotion Shell Application terminal window. Note that currently it is possible to access the X macro stack (push and pop) from the terminal.

Attributes:	**Type:**	Command.
	Axis related:	Yes.
	Array:	---.
	Assignment:	---.
	Receive parameter:	No.
	Parameter type:	---.
	Scope:	Communication.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

QN

Examples:

```
XQN     ‘ Reports the X macro program numbers stack.
YQN     ‘ Reports the Y macro program numbers stack.
```

See also:

QQ, QZ

15.4.16 QP – Macro Program Pointer

Purpose:

The Macro program pointer may be used to check the current program location, or to be assigned with a value (for indirect calls).

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No.
	Assignment:	Yes.
	Receive parameter:	---.
	Parameter type:	---.
	Scope:	Both.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	0 ÷ Max Macro Pointer.

Syntax:

QP

QP=<Number>

Examples:

XQP;	‘ Reports QP of X
XQP=1000; YQP=2000;	‘ Sets QP of X, Y respectively.

See also:

QE

15.4.17 QQ – Macro Program Stack

Purpose:

Reports the macro program subroutine's stack. The QQ keyword may be used to debug macro execution. The QQ command returns the subroutines call stack, in absolute macro pointers.

Attributes:	**Type:**	Command.
	Axis related:	Yes.
	Array:	---.
	Assignment:	---.
	Receive parameter:	No.
	Parameter type:	---.
	Scope:	Communication.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

QQ

Examples:

```
XQQ     ‘ Reports the X macro program subroutines stack.
YQQ     ‘ Reports the Y macro program subroutines stack.
```

See also:

QW

15.4.18 QR – Macro Initialization Status

Purpose:
Reports the macro initialization status. This is a bit array report only command.
Currently the following status bits are reported:

0x00000000	No Script is present.
0x00000001	Flag if downloaded macro encountered overflow
0x00000002	Internally Used
0x00000004	Flag if macro download finished
0x00000008	Flag if macro was downloaded successfully
0x00000010	Flag if macro was initialized successfully
0x00000020	Internally Used
0x00000040	Internally Used
0x00000080	Internally Used
0x00000100	Internally Used

Refer to the QF, for the macro running statuses.

Attributes:		
	Type:	Parameter.
	Axis related:	No.
	Array:	No.
	Assignment:	Zero ONLY !
	Receive parameter:	---.
	Parameter type:	---.
	Scope:	Both.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:
QR
Keyword is not axis related, thus XQR, YQR and BQR are equivalent.

Examples:
XQR ‘ Reports the Macro Status Register.
XQR=0 ‘ Clears Macro Initialized Flag.
See also: QI,QF

15.4.19 QT –Trace Macro Execution (Single Line)

Purpose:

Executes the relevant macro one clause at a time. This command is usually used in debugging mode. It is used by the Nanomotion Shell Application and debugging editor during macro programs debugging sessions.

Attributes:	**Type:**	Command.
	Axis related:	Yes.
	Array:	---.
	Assignment:	---.
	Receive parameter:	No.
	Parameter type:	---.
	Scope:	Both.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

QT

Examples:

```
XQT     ‘ Traces (executes) one clause from the X macro program.
YQT     ‘ Traces (executes) one clause from the Y macro program.
```

See also:

QE, QP

15.4.20 QU – Upload Macro Buffer

Purpose:
Uploads the macro program from the internal controller macro buffer, to the active communication link. This command is currently used by Using Nanomotion Shell application.

It may be used by the user from a stand-alone simple terminal only (will not work from the Nanomotion Shell Application terminal window).

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Receive parameter:	No.
	Parameter type:	---.
	Scope:	Communication.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:
QU

Keyword is not axis related, thus XQU, YQU and BQU are equivalent.

Examples:
```
XQU          ‘ Starts Macro Buffer upload.
```

See also:
QD, QL

15.4.21 QV – Uploads Descriptive Data

Purpose:

Uploads all macro descriptive data to the active communication line. The macro descriptive data includes the following information:

- Macro file name used to download the last program (downloaded file name and extension only, no full path).
- Last download date: format is DDMMYY.
- Descriptive comments (declared by the $description pre-compiler directive, each descriptive declaration in a separate line).

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Receive parameter:	No.
	Parameter type:	---.
	Scope:	Communication.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

QV

Keyword is not axis related, thus XQV, YQV and BQV are equivalent.

Examples:

```
BQV     ‘ Uploads the descriptive information.
```

See also:

$description directive.

15.4.22 QW – Wait till Condition

Purpose:

Waits for a specified internal state (variable). The desired state is provided as a parameter. For a list of supported internal states refer to section 10.8, Part III.

The QW command holds the macro execution until the state is satisfied.

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Receive parameter:	Yes.
	Parameter type:	Number.
	Scope:	Program.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

QW,<Constant>

Keyword is not axis related, thus XQW, YQW and BQW are equivalent.

Examples:

```
XQW,100000     ‘ Waits for the X axis to be in motion.
XQW,200000     ‘ Waits for the Y axis to be in motion.
XQW,200001     ‘ Waits for the Y axis to be not in motion.
```

See also:

QG

15.4.23 QZ – Clears Macro Numbers Stack

Purpose:
Clears the numbers stack. This command should be used when a new program starts running to avoid stack errors, in case previous functions did not leave the stack clear.

Attributes:	**Type:**	Command.
	Axis related:	Yes.
	Array:	---.
	Assignment:	---.
	Receive parameter:	No.
	Parameter type:	---.
	Scope:	Both.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:
QZ

Examples:
```
XQZ     ‘ Clears the X macro program numbers stack.
YQZ     ‘ Clears the Y macro program numbers stack.
```

See also:
QN

15.4.24 RT – Return from Subroutine

Purpose:
Returns from a subroutine call. Note that if the subroutine was not called using one of the Call Sub functions (CS, CF, CT) a stack error will occur.

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Receive parameter:	No.
	Parameter type:	---.
	Scope:	Program.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:
RT
Keyword is not axis related, thus XRT, YRT and BRT are equivalent.

Examples:

```
#MAIN:
XCS,#SUB_1 ‘Calls the SUB_1 Subroutine
XPA1=1     ‘ Return point of subroutine

#SUB_1:
XMO=1
XRT        ‘ Return from function
```

See also:
CS, CT, CF. QE

15.4.25 TD – Timer Down

Purpose:
The TD – Timer Down is a 32 bits timers (Axis related – one for each axis - positive only 0-2147000000) updated once this keyword is called. It is calculated according to the hardware interrupt entrance counter. When the timer reaches a value of '0' they stop.

Note: The values of the timer may be changed to any valid value from both communication and macro program.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	No
	Assignment:	Yes.
	Receive parameter:	---.
	Parameter type:	---.
	Scope:	Both.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	0.
	Range:	0 ÷ 2147000000.

Syntax:
iTD; iTD;
iTD=<Number>

Examples:
The following commands is a simple example for implementing a 1 second delay using XTD, and the **Timer (TD)** state condition:

XTD=8192; XQW,107000

Firstly, XTD is initialized, then the QW function is called, waiting for timer #1 to be zero.

See also:
QW, QG

15.4.26 ZA – Remote Assign Value (CAN Networking)

Purpose:

A Remote Assign parameter which sends an assignment clause to a remote unit. The parameter to be assigned is the command's parameter. The value to assign is taken from the numeric stack (one item removed).

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Receive parameter:	Yes.
	Parameter type:	String.
	Scope:	Program.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:
ZA,<String Parameter>

Keyword is not axis related, thus XZA, YZA and BZA are equivalent.

Examples:

```
XZI1=100          ‘ Remote Receive Address (RA=100)
XZI2=101          ‘ Remote Transmit Address (TA=101)
1};XZA,"XMO"   ‘ Remote Motor ON (MO=1)
```

See also:
ZC, ZI, ZM, ZR, ZS

15.4.27 ZC – Remote Command (CAN Networking)

Purpose:

A Remote Command which sends a command clause to the remote unit. The command to send is the ZC command's parameter. The command does not affect the numeric stack.

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Receive parameter:	Yes.
	Parameter type:	String.
	Scope:	Program.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

ZC,<String Parameter>

Keyword is not axis related, thus XZC, YZC and BZC are equivalent.

Examples:

```
XZI1=100        ‘ Remote Receive Address (RA=100)
XZI2=101        ‘ Remote Transmit Address (TA=101)
XZC,"XBG"       ‘ Remote Begin Motion (BG)
```

See also:

ZA, ZI, ZM, ZR, ZS

15.4.28 ZI – Remote Parameters Array (CAN Networking)

Purpose:

An axis related array parameter. The *i*ZI[1] array holds the address of the remote device to send messages to. The *i*ZI[2] array holds the address of the reply sent back to the FlexDC. *i* relates to the current macro.

Attributes:	**Type:**	Parameter.
	Axis related:	Yes.
	Array:	Yes (2 * 12)
	Assignment:	Yes.
	Receive parameter:	---.
	Parameter type:	---.
	Scope:	Both.
	Restrictions:	None.
	Save to Flash:	Yes.
	Default Value:	0.
	Range:	----.

Syntax:

XZI[1]; YZI[1];
XZI[1]= <Number>; YZI[1]= <Number>;

Examples:

```
XZI1=100        ‘ Remote Receive Address (RA=100)
XZI2=101        ‘ Remote Transmit Address (TA=101)
XZC,"XBG"       ‘ Remote Begin Motion (BG)
```

See also:

ZA, ZC, ZM, ZR, ZS

15.4.29 ZM – Remote Message (CAN Networking)

Purpose:

Sends a string message (limited to 8 characters) to a pre-defined remote, CAN ID address, defined by ZI[1] (ZI[2] is ignored). The message to send is the command's (string) parameter, or 1 or 2 or 3 numbers OFF number stack (number parameter).

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Receive parameter:	Yes.
	Parameter type:	String.
	Scope:	Program.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:

XZM,<string>

XZM,<Number (1 or 2 or 3) >

Examples:

XZM,"COMPLETE" Sends the COMPLETE word over the CAN bus to address XZI1

XZM,1 Sends the number at the top of the X stack (1 longs. Message size is 4 bytes).

XZM,2 Sends the two numbers at the top of the X stack. (2 longs Message size is 8 bytes)

XZM,3 Sends the three numbers at the top of the X stack. 2 bytes + 3 bytes + 3 bytes. Message size is 8 bytes. Top of stack is the 2 byte number, is set in the high bytes of the CAN message.

See also:

ZA, ZC, ZI, ZR, ZS

15.4.30 ZR – Remote Report Value (CAN Networking)

Purpose:
A Remote Report parameter which sends a report clause to a remote unit. The parameter to be reported is the command's parameter. The reported value is pushed to the numeric stack (one item is added to this stack).

Attributes:	**Type:**	Command.
	Axis related:	No.
	Array:	---.
	Assignment:	---.
	Receive parameter:	Yes.
	Parameter type:	String.
	Scope:	Program.
	Restrictions:	None.
	Save to Flash:	---.
	Default Value:	---.
	Range:	---.

Syntax:
ZR,<String Parameter>

Keyword is not axis related, thus XZR, YZR and BZR are equivalent.

Examples:
Remote access to a remote array element variable:

XZR, “XPA23”‘ Report remote XPA[23] (push one number to stack)

See also:
ZA, ZC, ZI, ZM, ZS

15.4.31 ZS – Remote Command Status (CAN Networking)

Purpose:
A parameter that holds the status of the last remote unit's response. Can only be reset to zero.

Attributes:	**Type:**	Parameter.
	Axis related:	No.
	Array:	No.
	Assignment:	No (read only).
	Receive parameter:	---.
	Parameter type:	---.
	Scope:	Both.
	Restrictions:	None.
	Save to Flash:	No.
	Default Value:	---.
	Range:	---.

ZS = 0 Last Message OK.
ZS = 1 Still Waiting for reply from remote unit.
ZS = 2 Remote Timeout or '?' returned from remote unit.
ZS = 3 Parameter syntax error.

Syntax:
ZS

Keyword is not axis related, thus XZS, YZS and BZS are equivalent.

See also:
ZA, ZC, ZI, ZM, ZR

Part IV – Nanomotion Shell Application

16 Introduction

16.1 General

Part IV covers the following FlexDC software applications: Nanomotion Shell Application and the Srcedit – the Macro File Editor Application.

These applications are applied for:

- Axis/axes online status reporting.
- User-friendly interface for controller's parameters.
- Easy axis/axes motion, for all motion modes.
- New firmware download.
- Macro download, upload and status reporting.
- Full IDE for macro editing and debugging.

17 Software Installation

The software installation consists of two stages:

- Hardware drivers' installation.
- Software installations.

17.1 Hardware Drivers' Installation

The FlexDC currently supports the following communication protocols:

- RS232 serial Communication Protocol
- CAN
- LAN

Each of these communication protocols needs a specific communication link.

The Nanomotion Shell Application program supports the following CAN cards:

- Kvaser CAN Cards.
- ESD CAN Cards.

These drivers installation are necessary (refer to the driver installation manuals supplied with the driver).

17.2 Software Installation

This section describes the installation of the FlexDC software and other utility programs.

17.2.1 Nanomotion Shell Application

Nanomotion Shell Application program is the GUI (Graphical Software Interface) for the FlexDC motion controller. This interface uses all controller keywords and enables easy access to the controller keywords and functionalities.

The FlexDC is supported by dedicated software, which requires a PC operating system with one of the following: Windows 98 / Windows 2000 / Windows XP. The software installation is user-friendly, using a dedicated Setup Utility provided on the product CD. The standard setup installation installs the FlexDC user manuals together with the Nanomotion Shell Application for user's convenience.

17.2.2 SCServer DCOM

The SCServer DCOM is a Hi-Level communication interface. This Hi-Level communication interface is used by the Nanomotion Shell Application, and is necessary for its operation. The "SCServerScope" is installed together with the Nanomotion Shell Application. Refer to Part V for further information.

17.2.3 SrcEdit Software

The SrcEdit is a powerful debugging environment and IDE for the FlexDC controller. The macro programs are integral part of the FlexDC. The debugging environment uses the low-level debug features of the FlexDC, while providing an advanced GUI support.

The SrcEdit is installed together with the Nanomotion Shell Application installation. Read further in Part IV for SrcEdit functionalities.

18 The Nanomotion Shell Application GUI

18.1 General

This chapter gives a detailed description of the Nanomotion Shell Application, its main screen (see Figure 20), available folders, dialogs and menus.

18.2 Main Screen

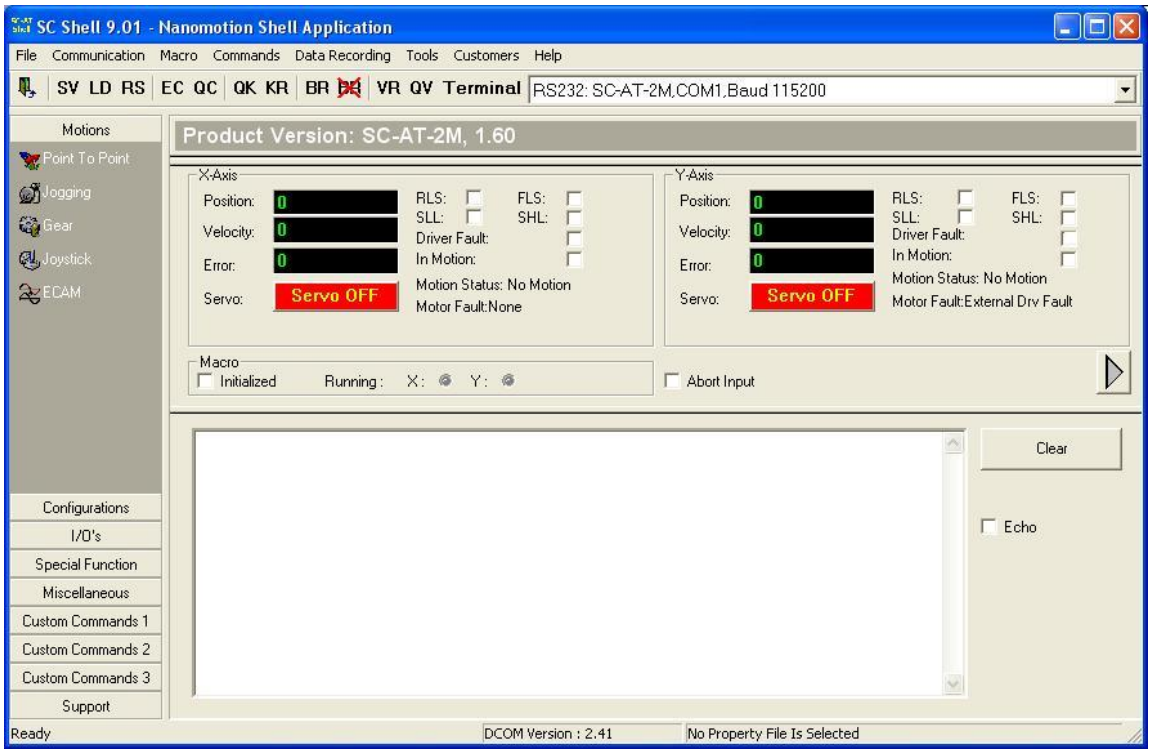


Figure 20: FlexDC Main Screen

The Main Screen Components:

- Selected Connection – enables the selection of defined connections.
- Version Control – current servo controller version.
- Status View – status of the main two axes and macro status per each axis.
- Folder View – current folder, selected by user.
- Outlook Bar – enables the selection of folders and custom commands.
- DCOM version – current DCOM server version the software used.

18.2.1 Axes Status Area

Axes status area gives a full status report on each axis (see Figure 21).

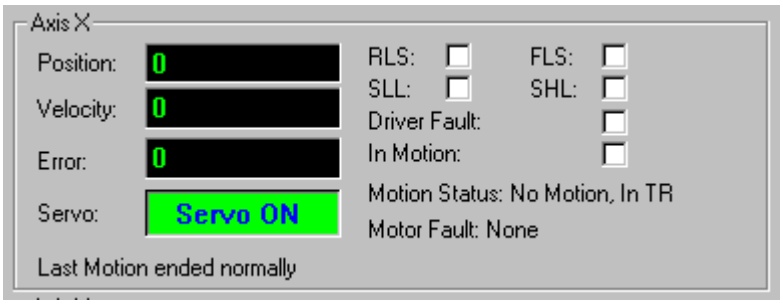


Figure 21: Axes Status Area

The Servo button – sets the servo ON or OFF. The status on the button states the current status of the controller. For instance, a green button with the *"Servo On"* text means that the controller axis servo status is currently on.

Note:

- *The status area view is updated at a rate of every 500 ms.*

The Axis Status Area shows:

- Position – current position.
- Velocity – current velocity.
- Error – position Error.
- Servo – Servo ON or Servo OFF.
- Last Motion End – last motion ending reason.
- RLS – reverse limit switch ON/OFF.
- FLS – forward limit switch ON/OFF.
- SLL – software low limit ON/OFF.
- SHL – software high limit ON/OFF.
- Driver Fault – driver Fault ON/OFF.
- In Motion –in motion ON/OFF.
- Motion Status – the current motion status description text.
- Motor Fault – text of motor fault, if one occurred.

18.2.2 Macro Status Area

The macro status area gives a status view of the currently running macro (see Figure 22).

Figure 22: Macro Status Area

The Macro Status Area reports the following:

- Initialized – If macro is initialized or not.
- Running – Status of each of the macros:
    - ♦ Green LED – macro is currently running.
    - ♦ Red LED – macro stop running due to an error.
    - ♦ Gray LED – macro currently is not running.

18.2.3 Version Control Area

The Version Control Area shows the current version of the servo controller and its communication parameters (see Figure 23).

Figure 23: Version Control Area

The version format is as follows:

- Servo controller type: SC-AT-2M (FlexDC).
- Firmware version.
- Communication name.
- Communication port.
- Communication baud rate.
- Servo controller location of connection. At a local computer or at a network computer.

18.2.4 Fast Menu Button

Figure 24: Fast Menu Button

The fast menu button (see Figure 24) shows a menu that includes:

- Show Terminal Window – changes the folder view to terminal view.
- Update Status Window – stops/resumes the main view automatic update.
- Watch Dialog – shows the watch dialog (see section 18.4.6.2, Part IV).

18.3 Folders

This section gives a detailed description of all the supported folders. The folders can be selected through the Outlook Bar, on the left side of the screen (see Figure 20).

All the folders include a Menu Button, see Figure 25:

Figure 25: The Menu Button

The Menu Button has two options:

- Download Parameters – pressing the left side of the Menu Button downloads all folder parameters to the FlexDC.
- Open folder's menu – pressing the right side of the Menu Button opens folder's specific menu. Most of the folders have a simple menu which enables:
  - Download Data To Controller – downloads folder's parameters to the FlexDC.
  - Upload Data From Controller – uploads folder's parameters from the FlexDC.
  - Set Default Values – sets folder's default values. This option does not download the data to the controller.

Notes:

- *Folders with Menu Button that include different options are fully described in this section.*

18.3.1 Motions Folder Group

Motions folder group includes folders for easy motion configuration and execution, see Figure 26:

Figure 26: Motion Folder Group

Each folder enables the following:

- Download, Upload and set default values to parameters.
- Start of an axis motion in the specified motion mode.

The Motions folder group includes the following folders[13];

- Point To Point motion folder – see section 18.3.1.1 (Part IV).
- Jogging motion folder – see section 18.3.1.2 (Part IV).
- Gear motion folder – see section 18.3.1.3 (Part IV).
- Joystick motion folder – see section 18.3.1.4 (Part IV).
- ECAM motion folder – current FlexDC firmware does not support this mode.

18.3.1.1 Point To Point Folder

Point-To-Point folder covers the Point-to-Point motion mode (MM=0), see Figure 27.

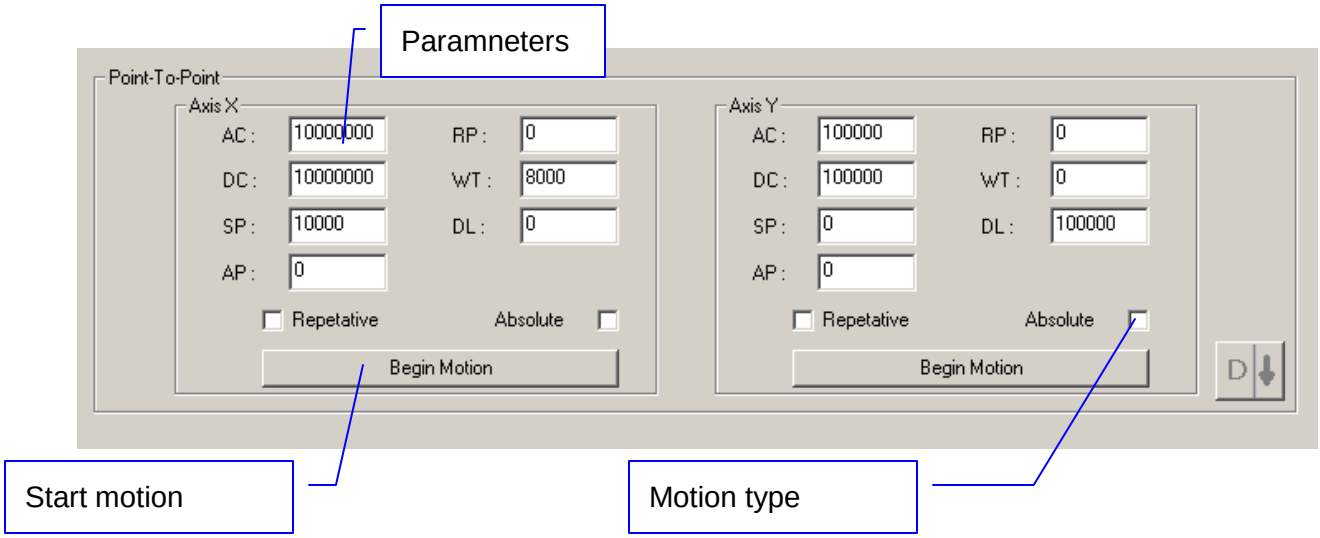


Figure 27: Point-To-Point Folder

[13] Each folder represents a different controller motion mode.

Point-To-Point folder parameters:

- AC – Acceleration.
- DC – Deceleration.
- SP – Speed.
- AP – Absolute position (desired absolute position).
- RP – Relative position (desired relative position).
- WT – Wait Period.
- DL – Limit Deceleration.
- Absolute – Absolute or Relative motion.
- Repetitive – Repetitive Motion Mode (SM=1).

The Begin Motion button downloads the axis parameters and starts the selected motion, presuming the relevant axis servo is ON.

Parameters can be updated while axis is in motion, to do so just change the relevant parameters and download them by pressing the Menu Button.

The following presents two examples:

The first one for starts an absolute motion and the other, starts an absolute repetitive motion.

1) Start absolute motion example:

1. Set the desired Acceleration parameter (AC).
2. Set the desired Deceleration parameter (DC).
3. Set the desired Speed (SP).
4. Set the desired Absolute position (AP).
5. Set the desired Wait Time (WT).
6. Set the desired Limit deceleration (DL).
7. Check the Absolute check box.
8. Uncheck the Repetitive check box.
9. Check that the axes servo is ON.
10. Press the Start Axis Button.

2) Start absolute motion that repeats itself example:

1. Set the desired Acceleration parameter (AC).
2. Set the desired Deceleration parameter (DC).
3. Set the desired Speed (SP).
4. Set the desired Absolute position (AP).
5. Set the desired Wait Period (WT).
6. Set the desired Limit deceleration (DL).
7. Check the Absolute check box.
8. Check the Repetitive check box.
9. Check that the Axes Servo is ON.
10. Press the Start Axis Button.

The same steps are performed for relative motions, with the following modifications:

- Clear the "Absolute" check box.
- Set the desired Relative position (RP).

18.3.1.2 Jogging Folder

Jogging Folder covers the Jog Motion Mode, see Figure 28.

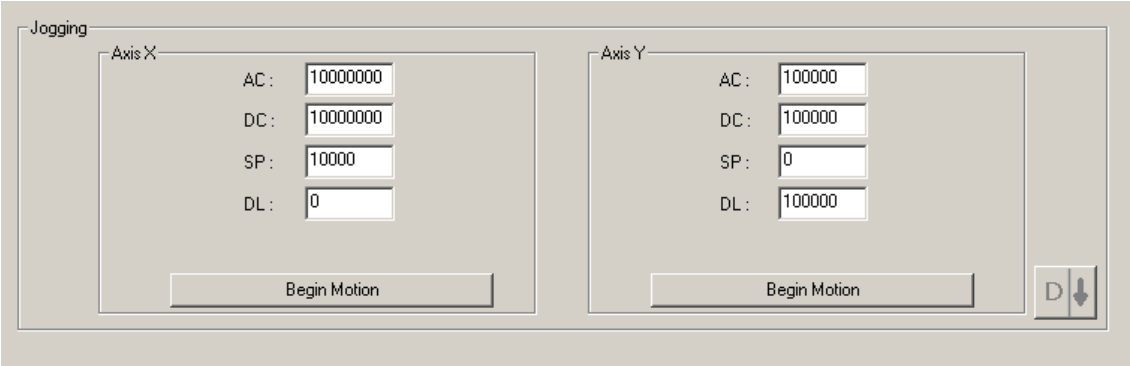


Figure 28: Jogging Folder

Jogging Folder parameters:

- AC – Acceleration.
- DC – Deceleration.
- SP – Speed.
- DL – Limit Deceleration.

Note:

- *The FlexDC supports normal jog mode only.*

The Begin Motion button downloads the axis parameters and starts the selected motion, presuming the relevant axis servo is ON.

The parameters can be updated while axis is in motion, to do so just change the relevant parameters and download them by pressing the Menu Button.

The following example starts a new jog motion:

1. Set the desired Acceleration parameter (AC).
2. Set the desired Deceleration parameter (DC).
3. Set the desired Speed (SP).
4. Set the desired Limit deceleration (DL).
5. Set jogging mode to Normal Jog mode.
6. Check that the axes servo is ON.
7. Press the Start Axis Button.

18.3.1.3 Gear Folder

Gear Folder covers the Gearing Motion Mode, see Figure 29.

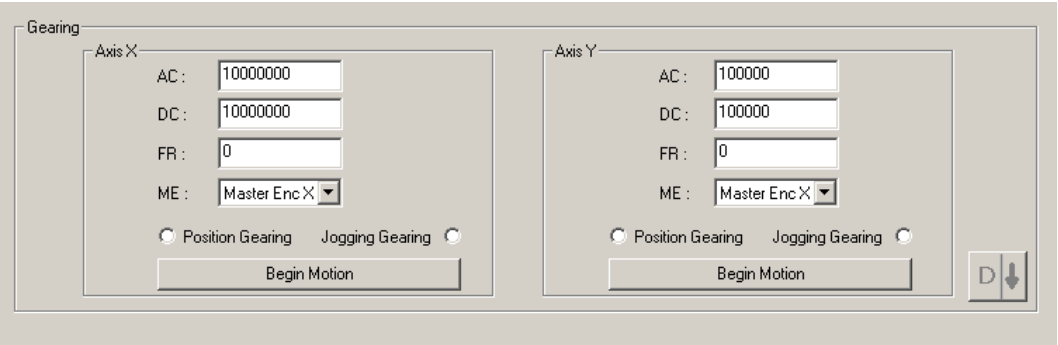


Figure 29: Gear Folder

Gear Folder parameters:

- AC – Acceleration.
- DC – Deceleration.
- FR – Master / Slave encoder ratio.

Gearing Mode:

- Position Gearing.
- Jogging Gearing.

The Begin Motion button downloads the axis parameters and starts the selected motion, presuming the relevant axis servo is ON.

Parameters can be updated while axis is in motion, to do so just change the relevant parameters and download them by pressing the Menu Button.

Next is an example for setting axis for gearing mode.

Start gear mode example:

1. Set the desired Acceleration parameter (AC).
2. Set the desired Deceleration parameter (DC).
3. Set the desired master slave encoder ratio (FR).
4. Set gearing mode to position mode.
5. Check that the axes servo is ON.
6. Press the Start Axis button.

18.3.1.4 Joystick Folder

Joystick Folder covers the Joystick Motion Mode, see Figure 30.

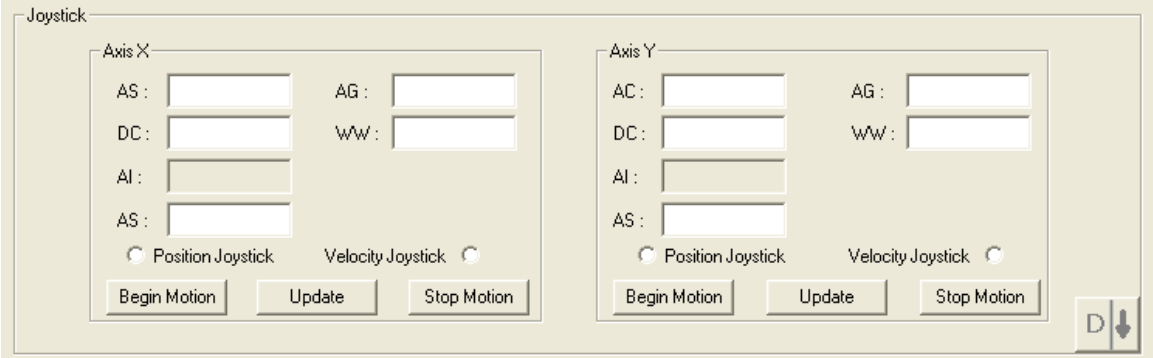


Figure 30: Joystick Folder

Joystick Folder parameters:

- AC – Acceleration.
- DC – Deceleration.
- AI – Analog Input (read only).
- AS – Analog offset.
- AG – Analog gain.

The Begin Motion button downloads the axis parameters and starts the joystick mode, presuming the relevant axis servo is ON.

Parameters can be updated while axis is in motion, to do so just change the relevant parameters and download them by pressing the Menu button.

Setting axis in joystick mode example:

1. Set the desired Acceleration parameter (AC).
2. Set the desired Deceleration parameter (DC).
3. Set Analog input offset (AS).
4. Set Analog input gain (AG).
5. Check that the axes servo is ON.
6. Press the Start Axis Button.

18.3.2 Configurations Folder Group

Figure 31: Configurations Folder Group

The Configurations folder group (see Figure 31) includes folders for configuring servo controller parameters.

Each folder enables the following:

- Download parameters to controller.
- Upload parameters from controller.
- Set parameters default values.

Configurations folder group includes the following folders:

- CAN folder – see section 18.3.2.1 (Part IV).
- Protection folder – see section 18.3.2.2 (Part IV).
- Configuration X, Y – see section 18.3.2.3 (Part IV).

18.3.2.1 CAN Folder

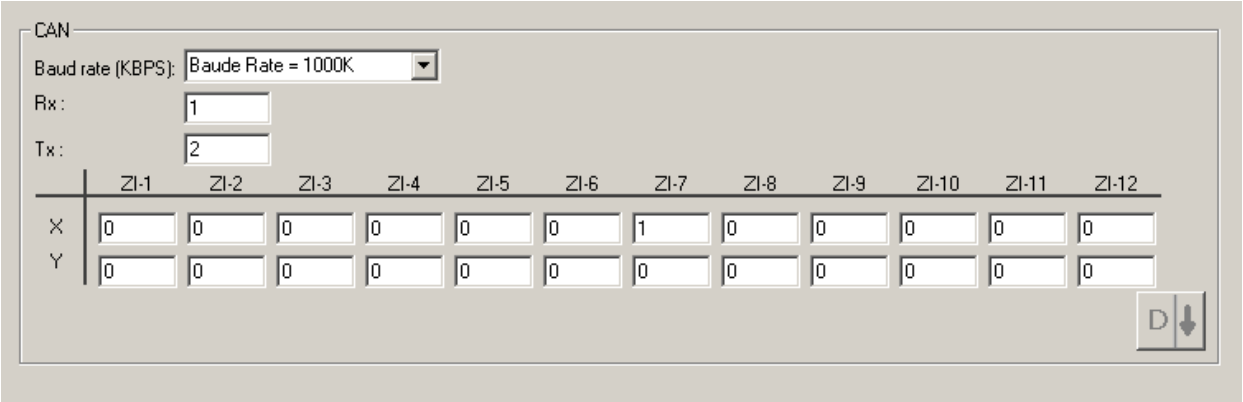


Figure 32: CAN Folder

CAN folder (see Figure 32) supports the following parameters:

- CAN baud rate.
- Rx address.
- Tx address.
- ZI arrays.

Changing the controller's CAN address:

1. Select the desired baud rate.
2. Enter the desired Rx address.
3. Enter the desired Tx address.
4. Enter ZI's values (optional).
5. Download the data to controller.

18.3.2.2 Protection Folder

The Protection folder (see Figure 33) includes the protection parameters, which are automatically loaded by the Nanomotion Shell Application.

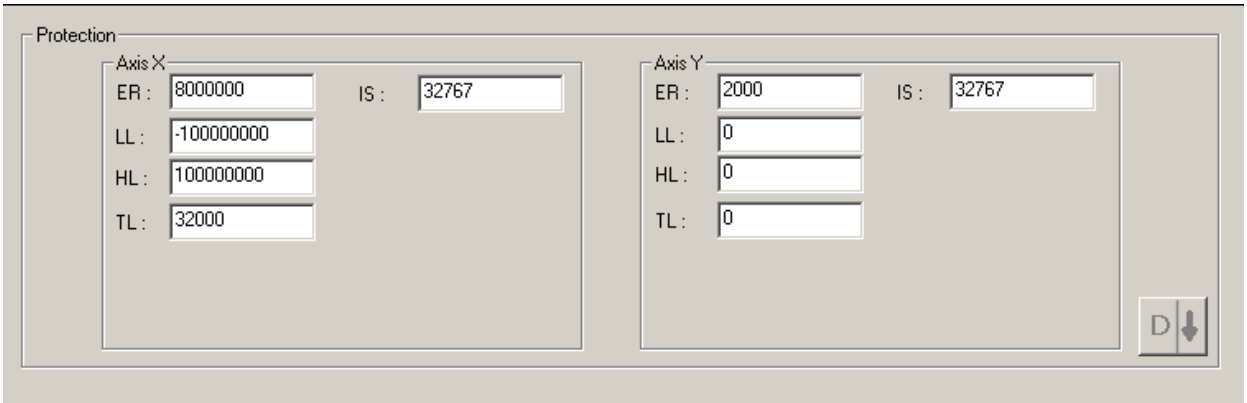


Figure 33: Protection Folder

Protection folder supports the following parameters:

- ER – Max position Error Limit.
- LL – the software Low Limit.
- HL – the software High Limit.
- TL – Torque Limit.
- IS – Integral Saturation limit.

Changing a protection parameter:

1. Change one or more of the protection parameters.
2. Download the data to controller.
3. Now the parameters are in the controller.
4. Save parameters to Flash Memory (SV).

18.3.2.3 Configuration X/Y Folders

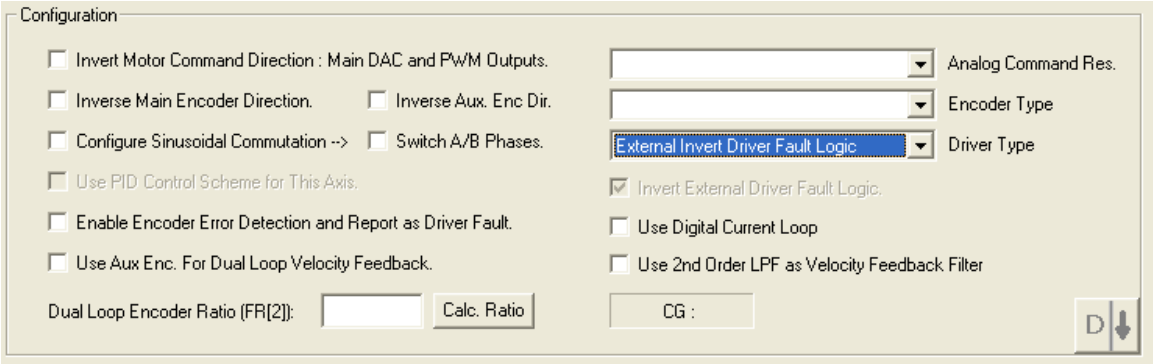


Figure 34: Configuration Folder

Configuration folder (see Figure 34) includes the following options, supported by the current FlexDC firmware:

- Invert motor command direction: main DAC and PWM output.
- Inverse main encoder direction.
- Analog command resolution 16 bit.
- Encoder type: A Quad B; Analog Sin/Cos.
- Driver Type: External Invert Driver Fault Logic.

Changing the configuration parameter:

1. Verify the status of the axis servo is OFF.
2. Change one or more of the configuration options.
3. Download the data to controller.
4. Save parameters to Flash Memory (SV).

18.3.3 I/O's Folder Group

I/O's folder group (see Figure 35) includes folders for configuring I/O's parameters and online status reporting.

Figure 35: I/O's Folder Group

Each of the I/O's folders enable:

- Download parameters to controller.
- Upload parameters from controller.
- Set parameters default values.

The I/O's folder group includes the following folders:

- I/O Logic folder – see section 18.3.3.1 (Part IV).
- Analog In folder – see section 18.3.3.2 (Part IV).
- Analog Output – see section 18.3.3.3 (Part IV).
- I/O Modes 0 – see section 18.3.3.4 (Part IV).

18.3.3.1 I/O Logic Folder

I/O Logic folder (see Figure 36) covers the logic and status of all the inputs and outputs in the controller.

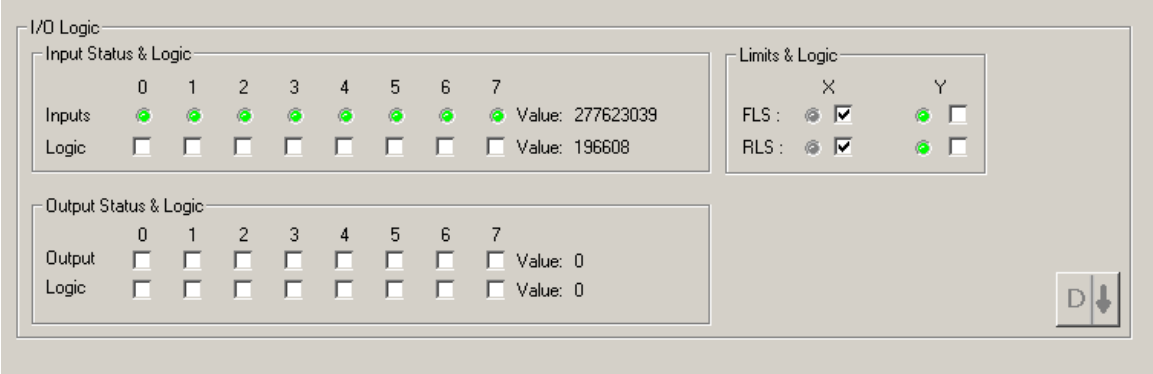


Figure 36: I/O Logic Folder

The I/O Logic folder is divided into three sections:

(1) Input Status & Logic – this section has two functionalities:

- Inputs – Shows the current status of the input. Green LED for ON, Red LED for OFF. The status is updated automatically. There is no need for uploading the parameters in order to see the updated inputs.
- Logic – Show/Set the current input logic. Setting input logic is done automatically once the user toggles the desired check box. The status of the input logic requires upload operation.

(2) Limits & Logic – this section handles the limits inputs status and logic according to the axis connections. The behavior of this section is the same as 'Input Status and Logic' section.

(3) Output Status & Logic - this section has two functionalities:

- Output – Show/Set the current output status. Setting output state is done automatically once the user toggles the desired check box. The status of outputs requires upload operation.

- Logic - Show/Set the current output logic. Setting output logic is done automatically once the user toggles the desired check box. The status of outputs logic requires upload operation.

18.3.3.2 Analog Input Folder

Analog Input folder (see Figure 37) covers the Analog input parameters.

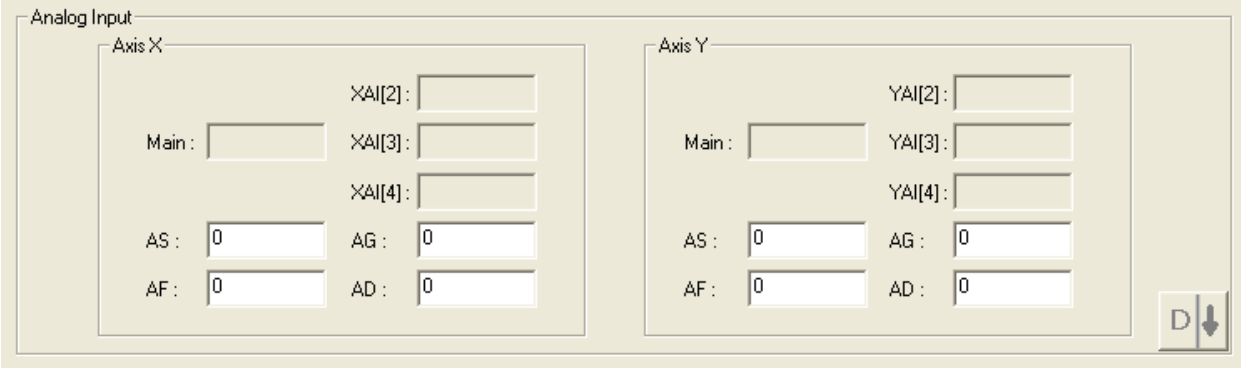


Figure 37: Analog In Folder

Analog Input folder supports the following parameters:

- XAI - YAI[1-4] – Analog input values (read only).
- AS – Analog offset (only for X, Y channels).
- AG - Analog gain (only for X, Y channels).
- AF – Analog input gain factor (only for X, Y channels).
- AD – Analog input dead band (only for X, Y channels).

The analog input value is automatically updated. The last two parameters are updated by pressing the Menu Button.

To change the analog input parameters:

1. Change one or more of the analog input parameters.
2. Download the data to controller.
3. Save parameters to Flash Memory (SV).

Note:

- *XAI – YAI [1-4] are read only parameters and as such it is not included in the download parameter option.*

18.3.3.3 Analog Out Folder

Analog Out folder (see Figure 38) covers the Analog output parameters.

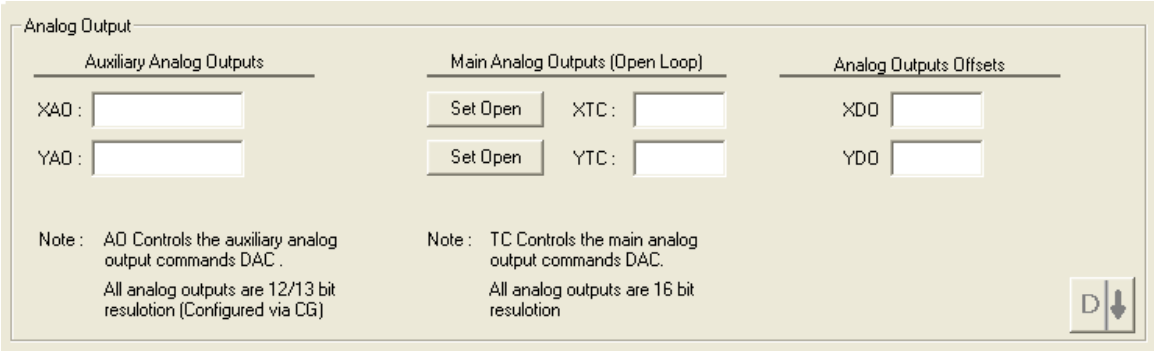


Figure 38: Analog Out Folder

Analog Out folder is divided into two sections:

(1) Analog Output parameters:

- XAO – X-axis analog output value.
- YAO – Y-axis analog output value.

(2) Open Loop parameters:

- XNC – X-axis enable/disable open loop.
- YNC – Y-axis enable/disable open loop.
- XTC – X-axis Torque (open loop) command.
- YTC – Y-axis Torque (open loop) command.

All folders parameters are downloaded/uploaded via the Menu Button only (see section 18.3, Part IV).

18.3.3.4 I/O Modes 0 Folder

I/O Modes 0 folder (see Figure 39) covers the control digital outputs (as normal or compare output function), and fast digital inputs parameters.

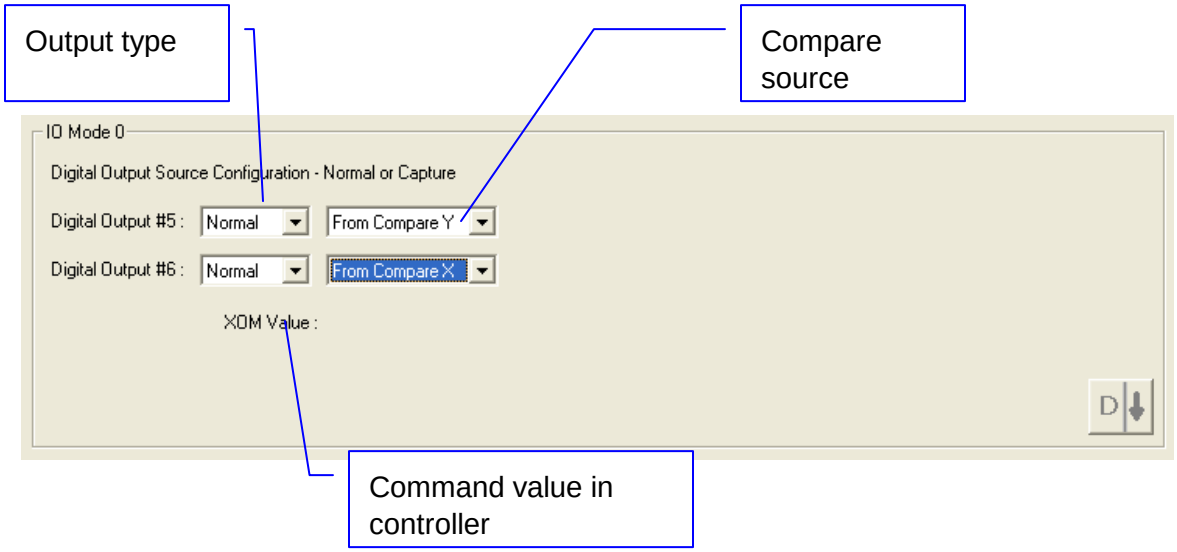


Figure 39: I/O Modes 0 Folder

I/O Modes 0 folder include the following options:

- Configure Digital Outputs enables two options:
  - Output Type - Normal or Compare mode.
  - Compare source to use, can be X or Y.
- Command value in controller – Shows the XOM command value currently in controller.

All folders parameters are downloaded/uploaded via the Menu Button only (see section 18.3, Part IV).

18.3.4 Special Function Folder Group

Special Function folder group (see Figure 40) includes folders for configuring special controller functions parameters and online status reporting.

Figure 40: Special Function Folder Group

Each of the Special Function folders enables (except to read only folders like Event Capture folder):

- Download parameters to controller.
- Upload parameters from controller.
- Set parameters default values.

Special Function folder group includes the following folders:

- Event Capture folder – see section 18.3.4.1 (Part IV).
- Event Generator X, Y, folders – see section 18.3.4.2 (Part IV).

18.3.4.1 Event Capture Folder

Event Capture folder (see Figure 41) covers the event capture controller variables status.

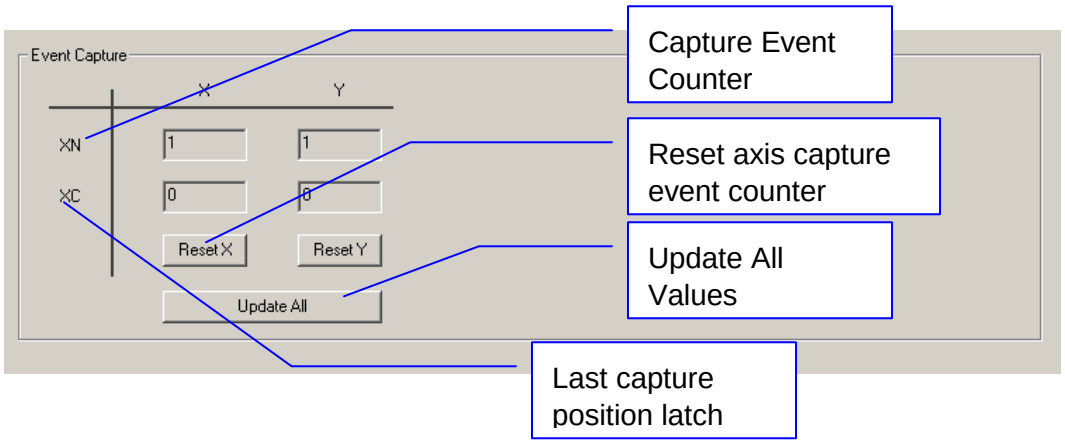


Figure 41: Event Capture Folder

The Event Capture folder enables three options for each axis:

- Capture Event Counter (XN) – Reports the number of capture events from the last capture events counter reset.
- Last Capture Position Latch (XC) – Reports the last capture location (position).
- Reset button – Clears the capture event counter for the axis (sends “XN=0” to controller).

In addition, the folders enable the upload of all folders parameters from controller via the "Update All Values" button.

18.3.4.2 Event Generator Folders

Event Generator folder (see Figure 42) covers the compare function parameters for axes X, Y.

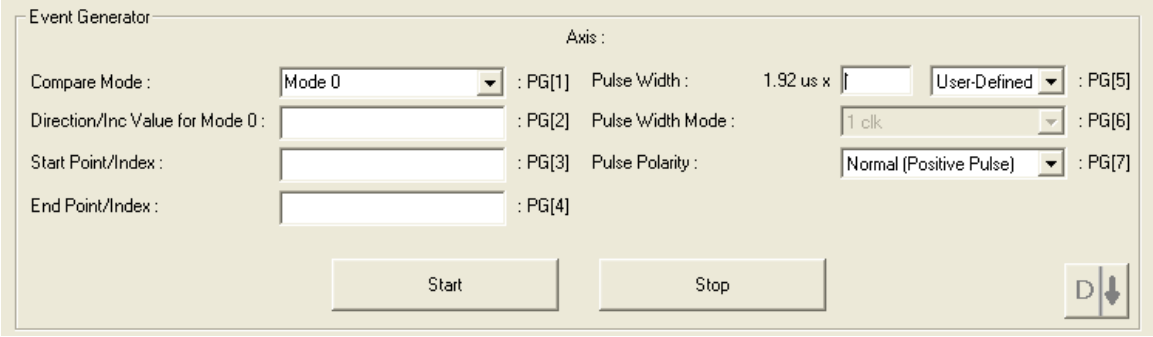


Figure 42: Event Generator Folder

Each Event Generator folder enables the following parameters editing/updating:

- Set the compare mode (PG[1]).
- Set the distance and direction according to the compare mode (PG[2]).
- Set the start position (PG[3]).
- Set the compare end point or index (PG[4]).
- Select the pulse width (PG[5]) – is selected, the user defined option defines a pulse with a width of 1.92us x *Value*.
- Select the pulse polarity (PG[7]).

In addition to the above the folder enables the following operations:

- Start button – Start current events generating (PQ,1).
- Stop button –Stop current events generating (PQ,0).

All folders parameters are manually downloaded/uploaded via the Menu Button only (see section 18.3, Part IV).

18.3.5 Miscellaneous Folder Group

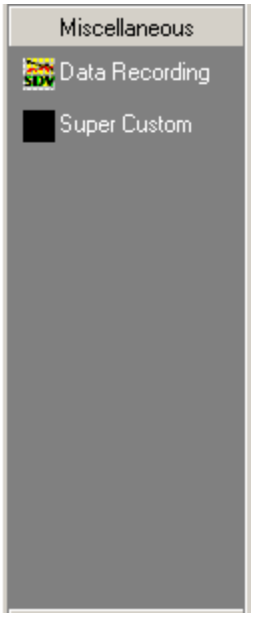


Figure 43: Miscellaneous Folder

Miscellaneous folder group (see Figure 43) includes the following folders:

- Data Recording – see section 18.3.5.1 (Part IV).
- Super Custom – see section 18.3.5.2 (Part IV).

18.3.5.1 Data Recording Folder

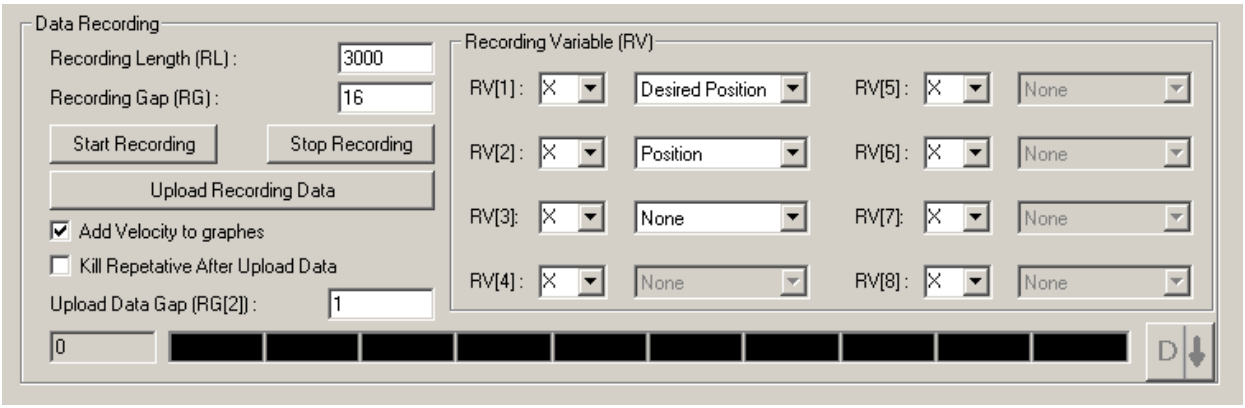


Figure 44: Data Recording Folder

Data Recording folders (see Figure 44) enables the control of all the recording parameters and gives an online status of current recording progress.

The folders parameters:

- Recording Length (RL) – number of points to record.
- Recording Gap (RG) – number of ISR between each recording point.
- Add Velocity to Graph – select this check box if it is required to add a software computed velocity to graph.
- Upload Data Gap – the gap between CAN messages when uploading recorded data.
- Kill Repetitive After Upload Data - select this check box if it is required to stop repetitive motion after the recorded data had been uploaded.
- Recording Variables (RV) – this UI section enables the selection of the variable to be recorded on relevant RV vector. Each vector can record every one of the two axes and every axis has the following possible recording data:
    - ♦ None.
    - ♦ Position.
    - ♦ Velocity.
    - ♦ Position Error.
    - ♦ Desired Position.
    - ♦ Driver Command.
    - ♦ Status Register.
    - ♦ Motion Status.
    - ♦ Analog Input.
    - ♦ Motor Fault.
    - ♦ Motor Status.
    - ♦ Motor's Current.
    - ♦ Filter Data Log.
    - ♦ Digital Inputs.
    - ♦ Digital Outputs.
    - ♦ Actual Digital Out.
    - ♦ Heatsink Temp.

The RV vectors must have sequential order this dictates the UI behavior. The UI will enable RV variable only if the RV before that has a value other than 'None'. If a RV variable value will be selected with 'None' value, all the RV that follows him will be reset to 'None' value and disabled.

The folder buttons:

- Start Recording – downloads all the recording parameters and starts the recording process.
- Stop Recording – stops the (current) ongoing recording.
- Upload Recording Data – uploads the recording data that is currently in the controller. This option enables the user to upload the recording data when required by the user.

After a recording is started the recording status is updated automatically in the 'Recording Status (RR)' variable.

A step-by-step example of recording 1000 points of X-axis position and X-axis Velocity:

1. Set Recording Length (RL) variable to 1000.
2. Set Recording Gap (RG) variable to 1.
3. Check the add velocity to graph, we add this option so we can compare between the actual velocity to the theoretical one.
4. Set Upload Data Gap to 1.
5. Set XRV/RV[1] combo boxes to 'X' and 'Position'.
6. Set YRV/RV[2] combo boxes to 'X' and 'Velocity'.
7. Push the Start Recording button.
8. The Recording Variable (RR) is automatically updated with the current recording status.
9. Once the recording is finished the shell will upload the recorded data automatically.
10. The SDV viewer will show the recorded data.

For more recording options see section 18.4.5, Part IV.

18.3.5.2 Super Custom Folder

Super Custom folder (see Figure 45) is a user-defined folder. The user can define the commands to be upload/download from/to controller and the give each command a user name.

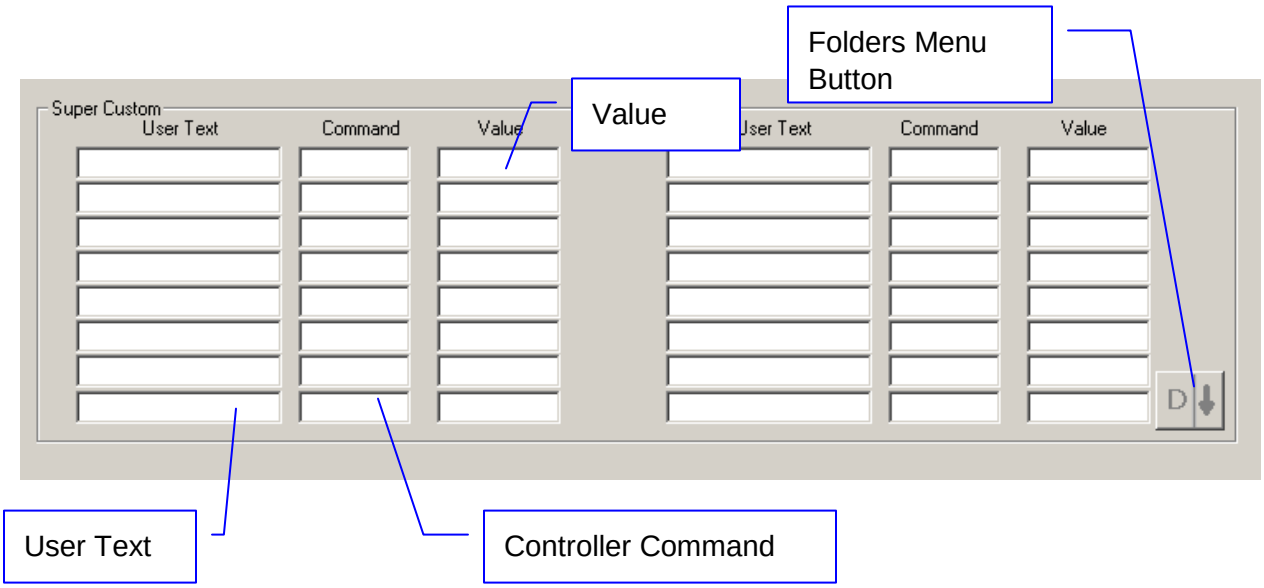


Figure 45: Super Custom Folder

Each user-defined variable has three values:

- User Text – a user can type any text.
- Command – the command to send to the controller. This command is one of the servo controller commands (refer to "Part II– FlexDC Software and Commands Reference").
- Value – the command value which can be the uploaded value from controller, or the downloaded value to the controller.

The Menu Button options:

- Download Data To Controller – downloads the commands and their values to controller.
- Upload Data From Controller – uploads the commands value from controller.
- Save Commands To File – saves all folders data to file.
- Load Commands From file – loads the folders data from file.

The folders data can be saved/loaded from/to file. This option enables the user to change between user-defined folders, for example have a user-defined folder for each of its customers

18.3.6 Custom Commands: 1 – 3 Folder Group

Custom commands (see Figure 46) are controller commands the user can send by a button click without the necessity of writing the command in the Terminal folder (see section 18.4.2.4, Part IV).

Custom commands are sent to the Terminal folder and therefore the reply is present at the Terminal folder.

The commands are automatically saved/loaded when the Nanomotion Shell Application is closed/opened and can be saved/loaded from file, for more information refer to 18.4.1, Part IV.

Custom command format can include more than one command separated by a "**;**" (semicolon).

Figure 46: Custom Commands Folder Group

Some examples for custom commands:

- XPS – Report X axis position.
- XPS;YPS – Report X and Y axis position.
- XPS=0;XVL=1000;XAC=1000;XDC=1000;XAP=10000;XSM=0;XMM=0;XMO=1;XBG – Set X axis parameters and begin motion.
- XQE,#HOME – Run X axis macro labeled HOME.

There are two options to edit a custom command: (one custom command at a time). Select the custom command to edit from the Outlook Bar, left mouse click on the custom command icon, and select the "Edit Custom" from the menu. The "Edit Custom Command" dialog box appears; see section 18.4.1.2, Part IV.

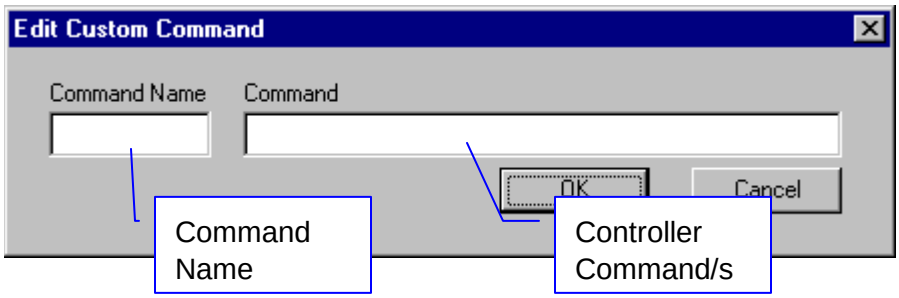


Figure 47: Edit Custom Command Dialog Box

The “Edit Custom Command” dialog box (see Figure 47) enables:

- Edit the command to be send via the “Controller Command/s” edit box.
- Edit the user text for the command. This text is shown at the Outlook Bar.
- Apply changes by pressing the OK button.
- Discard changes by pressing the Cancel button.
- All custom commands. To edit all commands in one dialog see section 18.4.1.2, Part IV.

18.3.7 Manuals Folder Group

The Manuals folder includes user manuals related to the FlexDC servo controller: the "FlexDC User Manual" and the "FlexDC Sofware User Manual" The manuals are updated from time to time and can be downloaded from the Nanomotion official web-site: www.nanomotion.com

18.4 Menus

This section gives a detailed description of all the menus (see Figure 48) the Nanomotion Shell Application currently supports.

Figure 48: The Nanomotion Shell Application Main Menu

18.4.1 File Menu

18.4.1.1 Load/Save Custom Commands

These menu items save/load custom commands to/from file.

To load custom commands from file follow the steps:

- On the "File" menu click the "Load Custom Commands".
- Choose the file to load from the "Load Custom Commands From" dialog box, the file must have ".scc" extension.
- Press the "Open" button to open the selected file.

Saving custom commands to file:

1. On the "File" menu click "Save Custom Commands".
2. Choose a file in the "Save Custom Commands To" dialog box, the file must have the ".scc" extension.
3. Press the "Save" button to save the selected file.

18.4.1.2 Edit Custom Commands

This menu item shows a dialog for editing all the custom commands at one dialog.

The "Edit Custom Command" (see Figure 49) dialog has two columns:

- Command Name – text that appears at the Outlook Bar.
- Command – custom command (refer to section 18.3.6) to be send. Each row in the dialog refers to one custom command.

The dialog buttons:

- "Last 15" button – browses the previous 15 custom commands.
- "Next 15" button – browses the next 15 custom commands.
- "OK" button – closes the dialog box and applies changes.
- "Cancel" button – closes the dialog box and discards changes.

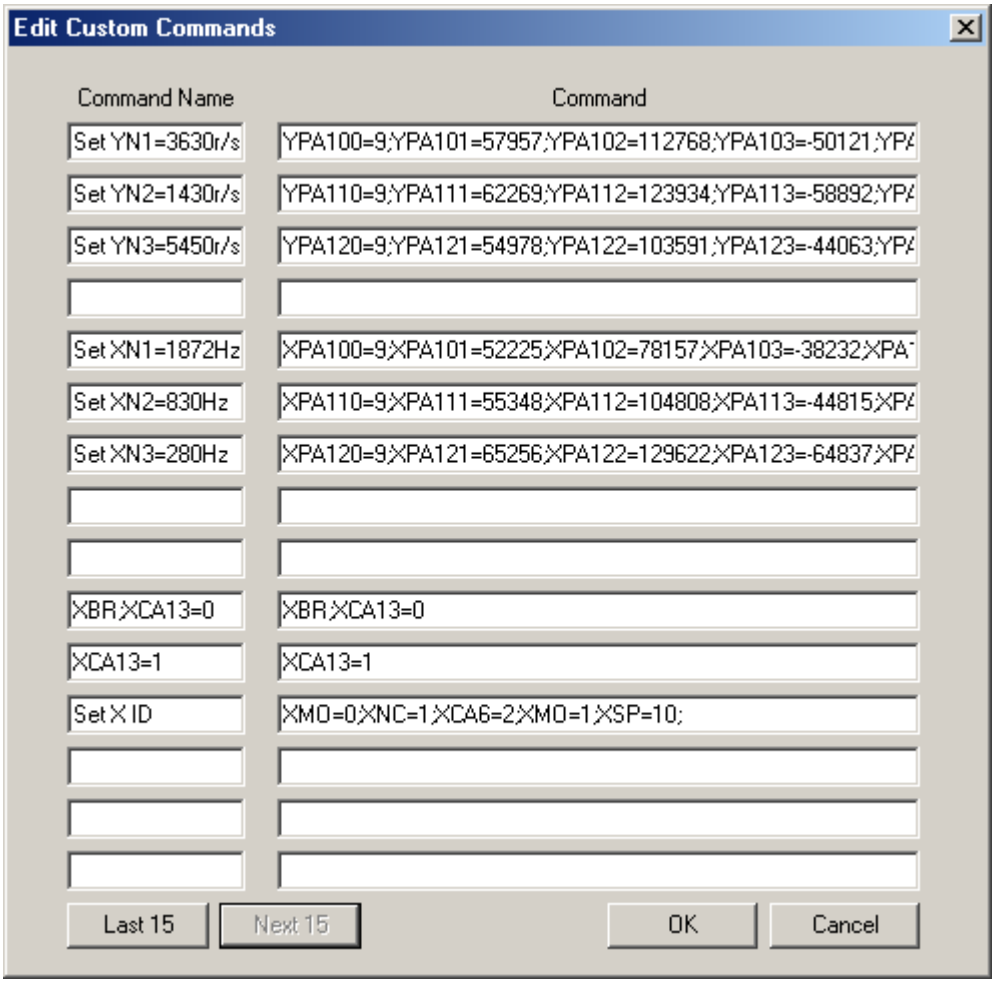


Figure 49: Edit All Custom Commands Dialog

18.4.1.3 File Location

The "File Location" menu shows the “File Location Setup” dialog box (see Figure 50).

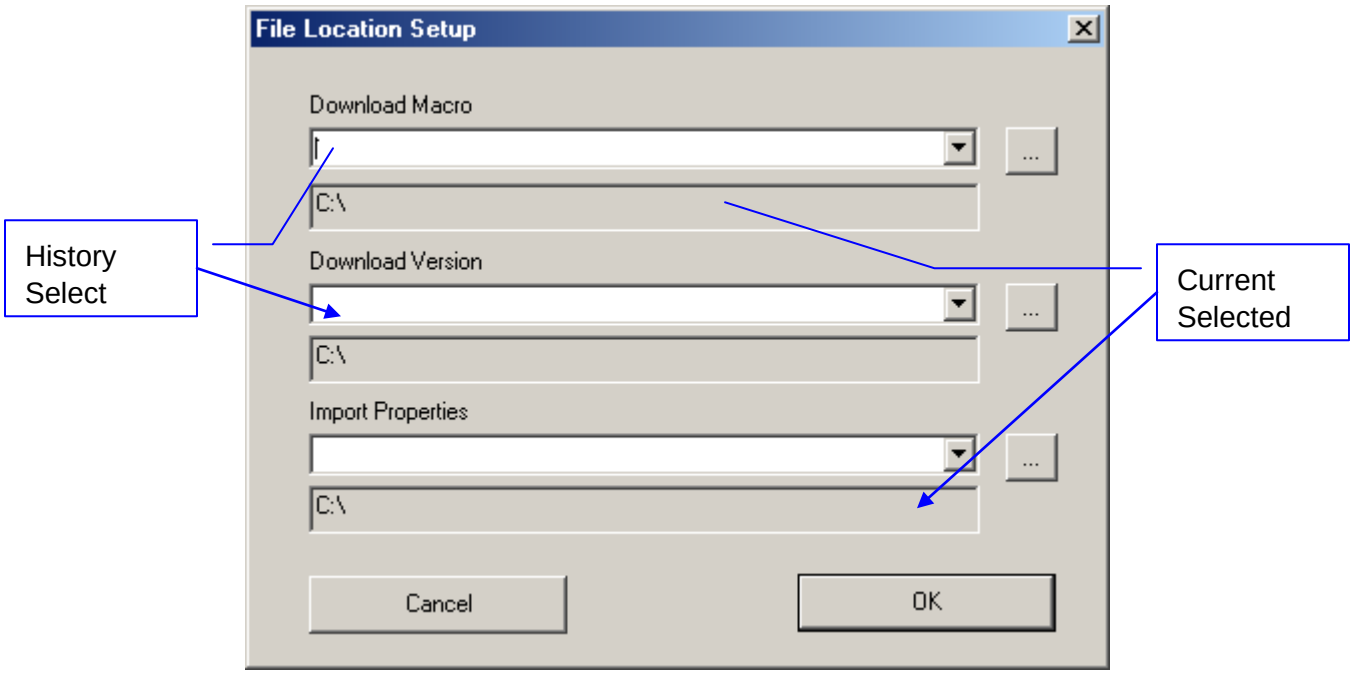


Figure 50: File Location Setup Dialog Box

This dialog box enables editing the file locations of:

- Download Macro – the location to open macro to be downloaded.
- Download Version – the location to open firmware version to be downloaded.
- Import Properties – the location and name of the recording properties file.

Every time the user chooses a macro or firmware version to download, the Nanomotion Shell Application opens the file browser at the location specified in this dialog.

The dialog components:

- Browse buttons - enable browsing the desired location.
- History Select – enables to select a location previously selected, from a combo box.
- Current Selected – shows the currently configured location.
- "OK" button – closes the dialog box and applies changes.
- "Cancel" button – closes the dialog box and discards changes.

18.4.2 Communication Menu

18.4.2.1 Setup Communication

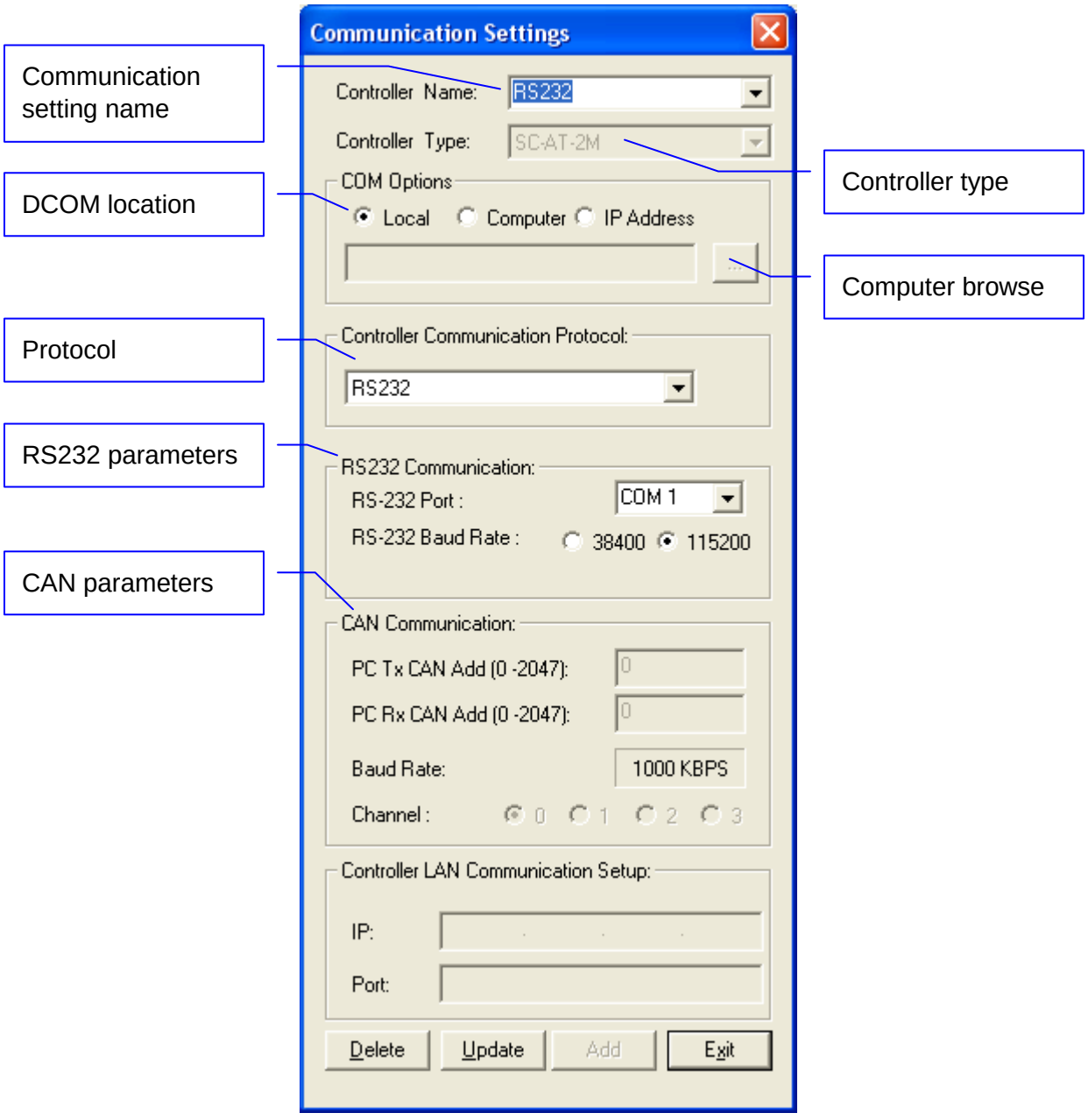


Figure 51: Communication Settings Dialog Box

The "Setup Communication" menu (see Figure 51) enables the communication type definition for using the controller.

The Setup Communication dialog box components:

- Controller Name – user defined communication setting name. This name is unique to each type of communication.
- Controller Type –the supported controller is SC-AT-2M (FlexDC).
- DCOM Location – the location of DCOM server. The location defines the computer, the controller is connected to. The location can be:
  - Local – on the current computer.
  - Computer – a computer name on the local area network (use the "Browse" button under the "COM Options" for a list of all computers on the local area network).
  - IP Address – a computer IP address.
- Controller Communication Protocol – communication protocol can be RS232 or CAN.
- RS232 Communication – parameters for the RS232 communication, the parameters list:
  - RS232 Port – COM port the controller is connected to.
  - RS232 Baud Rate – RS232 Connection speed (the current FlexDC firmware supports 38400 bps).
- CAN Communication Parameters – parameters for the CAN communication, the parameters list:
  - PC Tx CAN Add (0-2047) – PC CAN transmit address.
  - PC Rx CAN Add (0-2047) – PC CAN receive address.
  - Baud Rate – CAN communication speed, this combo-box is read only, the DCOM server defines the speed (refer to "Part V– SCServer COM/DCOM Interface Library").
  - Channel – the CAN channel, currently supported:
    - ► Kvaser cards – up to four channels, depending on the CAN hardware card installed.

- ► ESD cards – only one channel supported.

- Dialog box buttons:
  - "Delete" – deletes the current selected communication definition. This option also deletes all the views that are defined with this communication.
  - "Update" – updates the communication definition.
  - "Add" – adds the current setup to the communication list, error messages are given for missing information and an identical communication name.
  - "Exit" – closes the dialog box.

18.4.2.2 Update Status Window

On "Communication" menu, click the "Update Status Window" menu item. This menu item stops or resumes the Nanomotion Shell Application main screen update. Clicking this menu item, updates the main screen twice a second. Using this menu item, the user can send commands via the Terminal folder (see section 18.4.2.4, Part IV).

18.4.2.3 Connect/Disconnect Last Selected

- The "Connect Last Selected" menu item connects to controller with the last successful connection.
- The "Disconnect Last Selected" menu item disconnects current connection.

The last successful connection is saved/load when the Nanomotion Shell Application is closed/opened, so this menu item is available even after the shell is closed or open.

18.4.2.4 Show Terminal Window

The "Show Terminal Window" menu item changes the current selected folder to the Terminal Folder (see Figure 52.

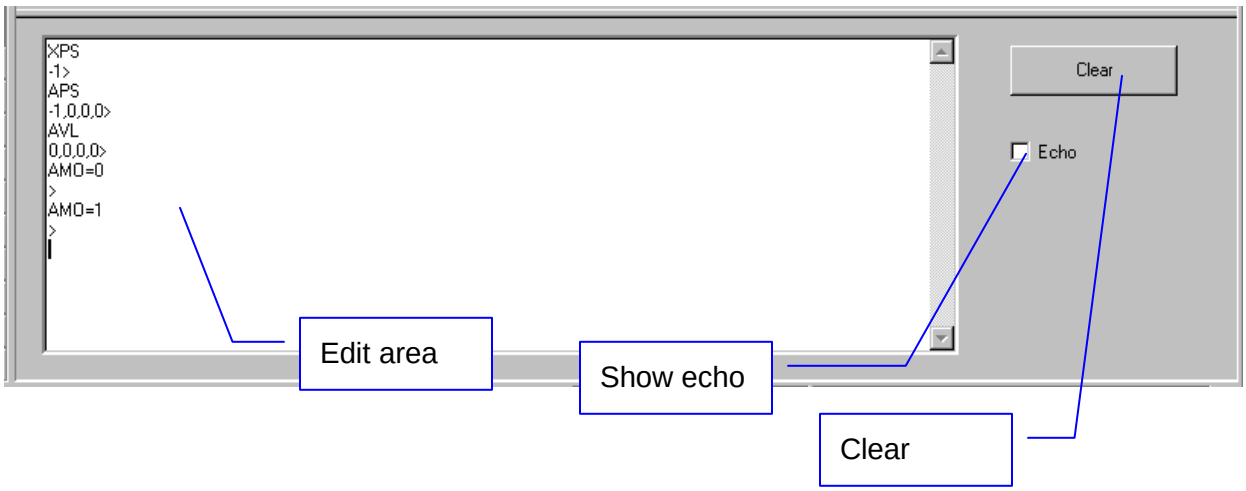


Figure 52: Terminal Folder

The Terminal window's purpose is to send commands to the controller and to show the controller's responses (Custom commands are also sent via the Terminal window).

Terminal folder components:

- "Echo" check box – enables/disables echo of the command/s sent to the controller. For example - the sent to the controller command "XPS" has the answer "XPS=10000" when the "Echo" is enabled and "1000" if the "Echo" is disabled.
- "Clear" button – clears the edit area from text and the history commands (see next component).

- Edit Area – area in which the user writes the commands and reads controller's answers. This area has the following edit features:
  - Edit commands with all alphabetic chars.
  - Enter for sending the command/s to controller.
  - Backspace key to delete and edit the command.
  - Up/Down arrow keys to browse through previous commands sent.
- Terminal command format can include more than one command separated by ';' (semicolon).

18.4.2.5 Show Previous Folder

Show Previous Folder menu item switch between the current folder and the previous folder.

18.4.3 Macro Menu

18.4.3.1 Pre-compile Macro

The "Pre-Compile" menu item compiles a given macro without downloading it to the controller.

Pre-compile a macro step-by-step:

1. On "Macro" menu click the "Pre-Compile" menu item.
2. Choose the macro file to compile from the "Pre-Comple Macro File" dialog box. Note: the macro files have the ".scm" extension.
3. A process progress dialog box shows the pre-compilation status.
4. When the pre-compilation is finished a message box appears with the compilation result.

18.4.3.2 Download Macro

The "Download Macro" menu item pre-compiles and downloads a given macro file.

Download macro step-by-step:

1. On the "Macro" menu click the "Download Macro" menu item.
2. Choose the macro file to compile and download. Note, the macro files have the ".scm" extension.
3. A process progress dialog box shows the download status.
4. When the download is finished a message box appears with the download result.
5. Save parameters to Flash Memory, if needed.

18.4.3.3 Download .DAT File

The "Download DAT File" menu item downloads a compiled macro file. The .DAT file is the pre-compile file generated for download.

Download .DAT file step-by-step:

1. On the "Macro" menu click the "Download DAT File" menu item.
2. Choose the .DAT file to download. Note, that the .DAT files have the ".dat" extension.
3. A process progress dialog box shows the download status.
4. When the download is finished a message box appears with the download result.
5. Save parameters to Flash Memory, if needed.

18.4.3.4 Upload Macro

The "Upload Macro" menu item uploads macro from controller. The file created by this process is a .DAT file, the same .DAT file created by the macro pre-compile process, and can be downloaded by "Download DAT File" menu item, see section 18.4.3.3, Part IV.

Upload Macro step-by-step:

1. On the "Macro" menu click "Upload Macro" menu item.
2. Choose the DAT file to save the macro to. Note, the file should have the ".dat" extension.
3. A process progress dialog box shows the upload is in progress status.
4. When the upload is finished a message box appears with the upload results.

18.4.3.5 Erase Macro

The "Erase Macro" menu item erases the macro from controller. Save parameters to Flash Memory, if needed.

18.4.4 Commands Menu

18.4.4.1 Save

The "Save" command saves controller's parameters to the Flash Memory (SV).

18.4.4.2 Load

The "Load" command loads controller's parameters from the Flash Memory (LD).

18.4.4.3 Reset Controller

The "Reset Controller" command resets the controller (RS).

18.4.4.4 EC

The EC command shows the communication error code text (EC).

18.4.4.5 Show RTE

The "Show RTE" command shows all axis macro runtime error (QC).

18.4.4.6 Kill All

The "Kill All" command kills all running programs.

18.4.4.7 Kill Repetitive

The "Kill Repetitive" command kills all axis repetitive motions.

18.4.5 Data Recording Menu

18.4.5.1 Import Properties

The "Import Properties" menu item imports the SD viewer application properties for recorded data.

18.4.5.2 Clear Properties

The "Clear Properties" menu item clears properties imported by "Import Properties" menu item.

18.4.5.3 Show Last Recording

The "Show Last Recording" menu item opens the SD viewer with the last recorded data. This menu item does not upload the recorded data.

18.4.5.4 Open Graph Application

The "Open Graph Application" opens the SD viewer.

18.4.5.5 Start Recording with Begin Motion

The "Start Recording with Begin Motion" starts the recording when one of the axes receives the Begin Motion command. Select the "Start Recording with Begin Motion" to start recording with BG command. Deselect this menu item and the BG command does not start the data recording.

The recording parameters must be downloaded to controller before the begin motion occurs.

18.4.5.6 Stop Current Recording Process

The "Stop Current Recording Process" menu item stops the current ongoing recording process.

This menu item does not upload the recorded data.

18.4.5.7 Start Recording

The "Start Recording" sends a start recording command to the controller.

18.4.6 Tools Menu

This menu includes Tools for easy array editing and watch dialog to view online changes in the controller.

18.4.6.1 Array Editing

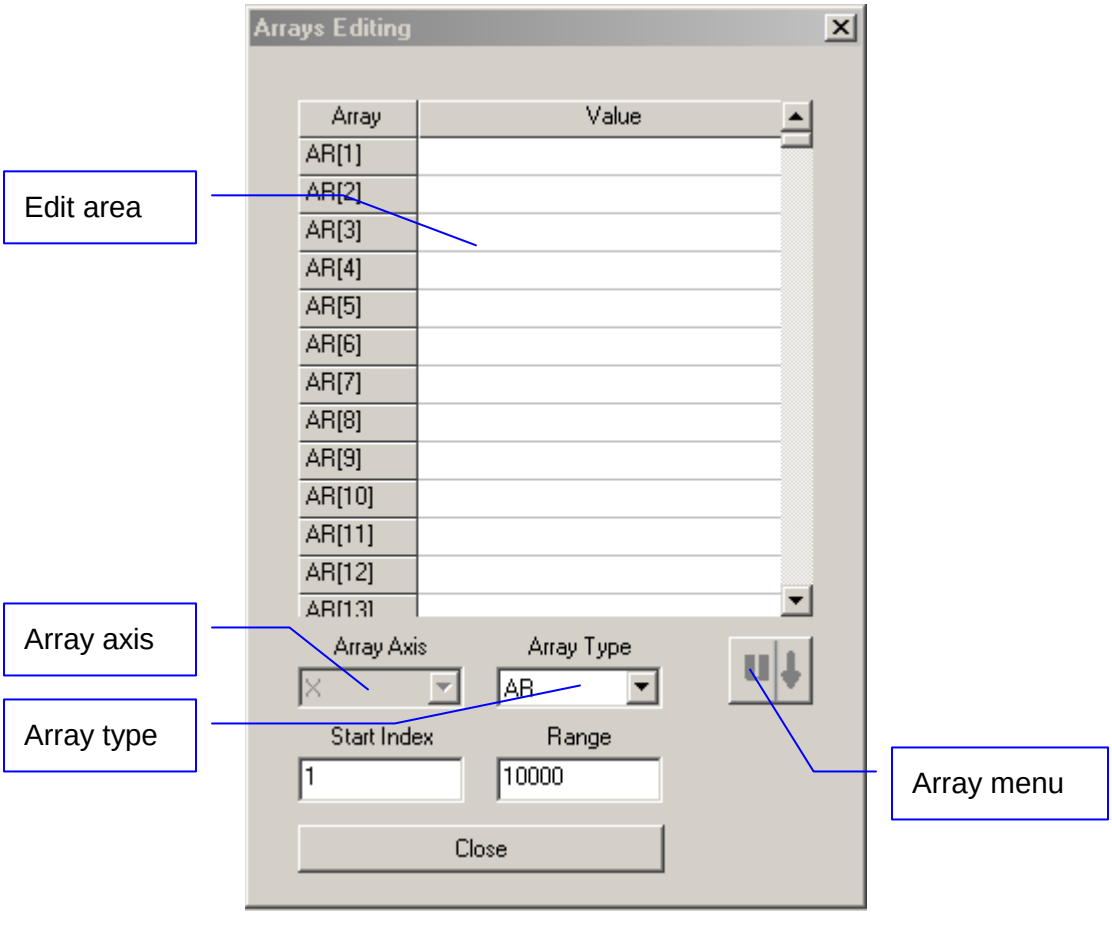


Figure 53: Array Editing Dialog

The "Array Editing" menu item enables editing controller's arrays (see Figure 53).

To edit an array:

1. Choose the array type to edit from the "Array Type" combo-box.
2. If the array is axis related the dialog enables the selection of the desired axis from the "Array Axis" combo-box.
3. Modify the array directly in the "Edit Area" dialog box or use one of the options in the Array Menu (see menu items below).
4. At any stage values can be modified via the "Edit Area" dialog box.

The Array Menu dialog box includes:

- Load Array From File – loads an array values from file. The range is defined on the dialog start and range edit box.
- Save Array To File – loads an array values to file. The range is defined on the dialog start and range edit box. Only values that are not empty are saved.
- Upload Array From Controller – uploads an array values from controller. The range is defined on the dialog start and range edit box.
- Download Array to Controller – downloads an array values to controller. The range is defined on the dialog start and range edit box.

The user can use sequence of array operations, this way the user can build an array from the controller values and file values, for example:

A certain file has AR values from index 1 to 100, and another file with AR values from index 200-210. The user can upload AR values twice: once for index 1-100 and once again for index 200-300. Then, use the menu item "Download Array to controller" to download the values to the controller.

18.4.6.2 Watch Dialog

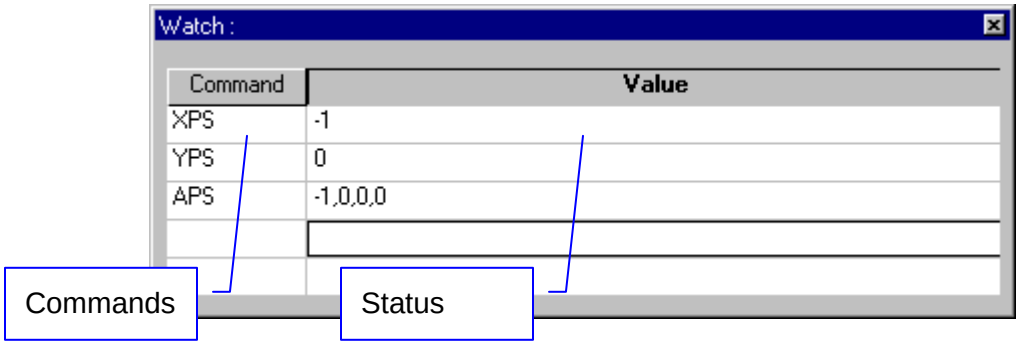


Figure 54: Watch Dialog

The Watch Dialog menu shows a floating dialog box that shows commands' status.

The Watch Dialog box (see Figure 54) has two parts:

- At the left side are the user's commands (any command that can be read from the controller, except upload macro and recorded data commands).
- At the right side are the commands' values which are retrieved from controller; commands values are updated at the same time the main screen is updated.

Edit a command step-by-step:

1. Select the cell to edit, at the grids left side.
2. Enter the desired command, for example XPS.
3. When finished editing the cell press enter.

To edit another command repeat the above process.

19 "Srcedit" – the Macro File Editor Application

19.1 General

"Srcedit" is the Macro File Editor Application. Its purpose is to enable the user to edit and debug controller's macros.

The editor has two working modes:

- Edit mode – in this mode, the Macro File Editor Application works as a text editor with macro command syntax coloring (for more details see section 19.4, Part IV).
- Debug mode – in this mode, the Macro File Editor Application works as a code debugger that enables running the macro, stopping it, step-by-step running, breakpoints etc… (for more details see section 19.6, Part IV).

For macro syntax refer to "Part III– FlexDC Macro Language".

19.2 Main Screen

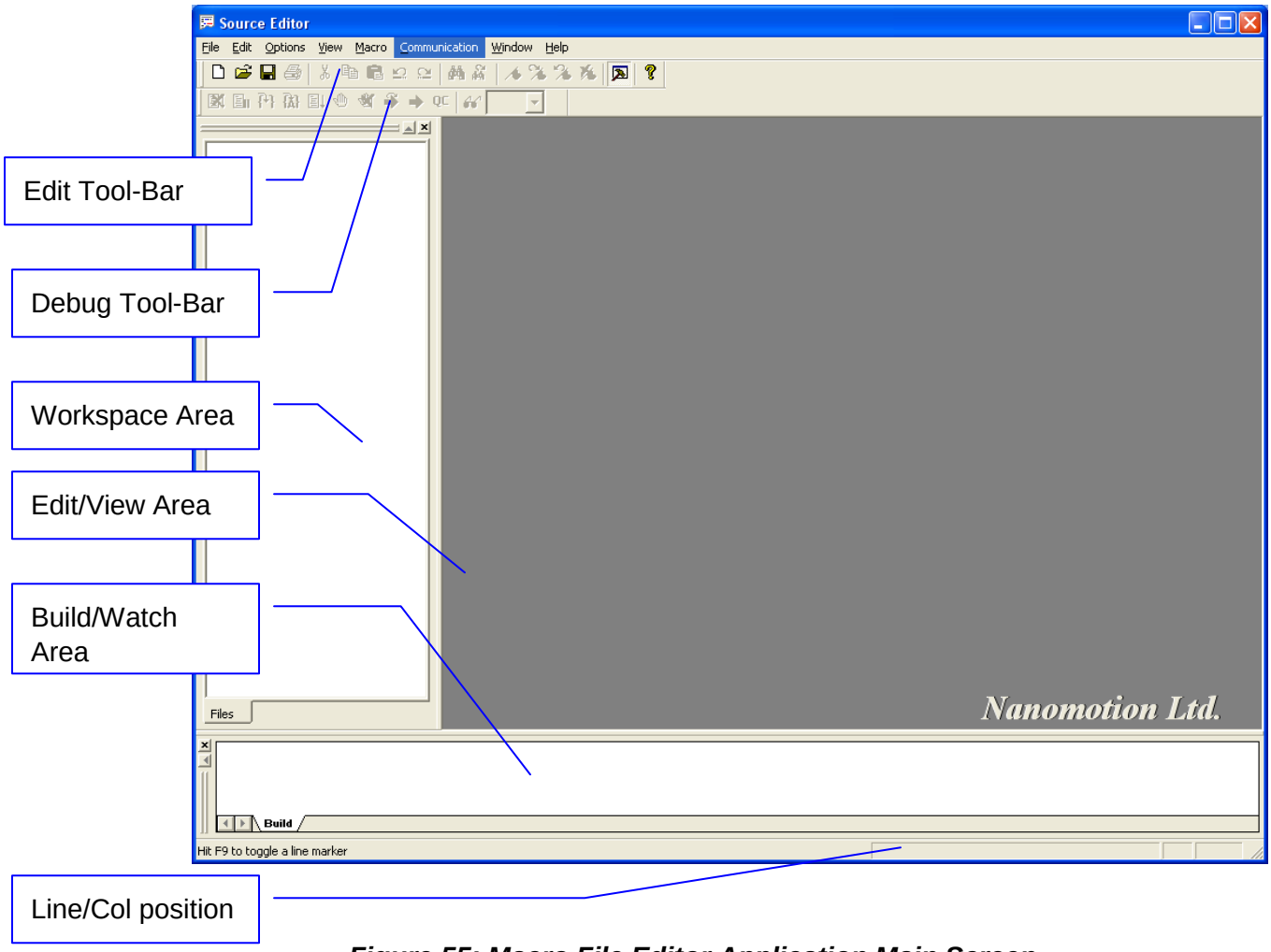


Figure 55: Macro File Editor Application Main Screen

This section gives a short description of the Macro File Editor Application main screen (see Figure 55) components.

Macro File Editor Application Main screen components:

- Edit Tool-Bar – for editing options, enabled only in edit mode.
- Debug Tool-Bar – for debugging options, enabled only in debug mode.
- Workspace Area – shows the current files included in the workspace (see section 19.3, Part IV).

- Edit/View Area – this area enables:
  - Editing macro file in edit mode (see section 19.4, Part IV).
  - Show debugging status and options in debug mode (see section 19.6, Part IV).
- Build/Watch Area – shows the results of macro compiling and used as a Watch View in debug mode.

19.3 Workspace

Workspace purpose is to help the user manage macros that include more than one file. For example, a macro file can include two files: one is a definitions file and one is of the actual code. The best way to manage this macro file is working with workspace, see Figure 56.

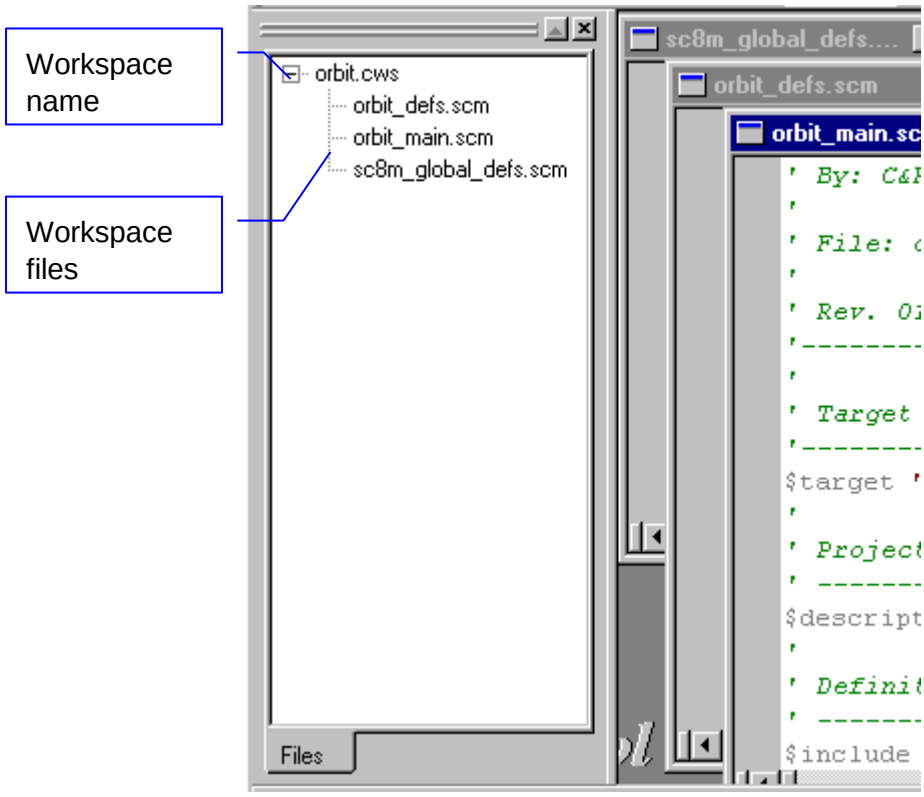


Figure 56: Workspace Area

Workspace operations:

- Creating a new workspace – on the "File" menu click "Workspace", click "New Workspace", select or create a new workspace file. A new workspace name will appear at the workspace area.
- Open/Close/Save workspace – on the "File" menu click "Workspace", click "Save" (or "Save As") to save a workspace; or click "Open" to open a workspace; or click "Close" to close a workspace.
- Adding files – there are two ways to add a file to workspace:
  - Drag-and-drop a file into the workspace.
  - On the "File" menu click "Workspace", click "Add File" and select a file from the dialog box.
- Deleting files – there are two ways to delete a file from workspace:
  - Select a file in the workspace, left mouse click and select "Remove File".
  - Select a file in the workspace. On the "File" menu click "Workspace", click "Remove File".
- Compiling/Debugging workspace – to compile a workspace the main macro file should be opened and on top of all other opened files. Compiling is performed on the top most opened file (for more information on macro compiling and downloading, see section 19.7.5, Part IV).
- Open file in Edit/View Area – There are two ways to open file in workspace:
  - Double click on the file in the workspace area.
  - On the "File" menu click "Workspace", press "Open" and select a file.

Note:

- *Workspace changes are saved only after saving the workspace, closing the workspace without saving discards the changes*

19.4 Macro Editing

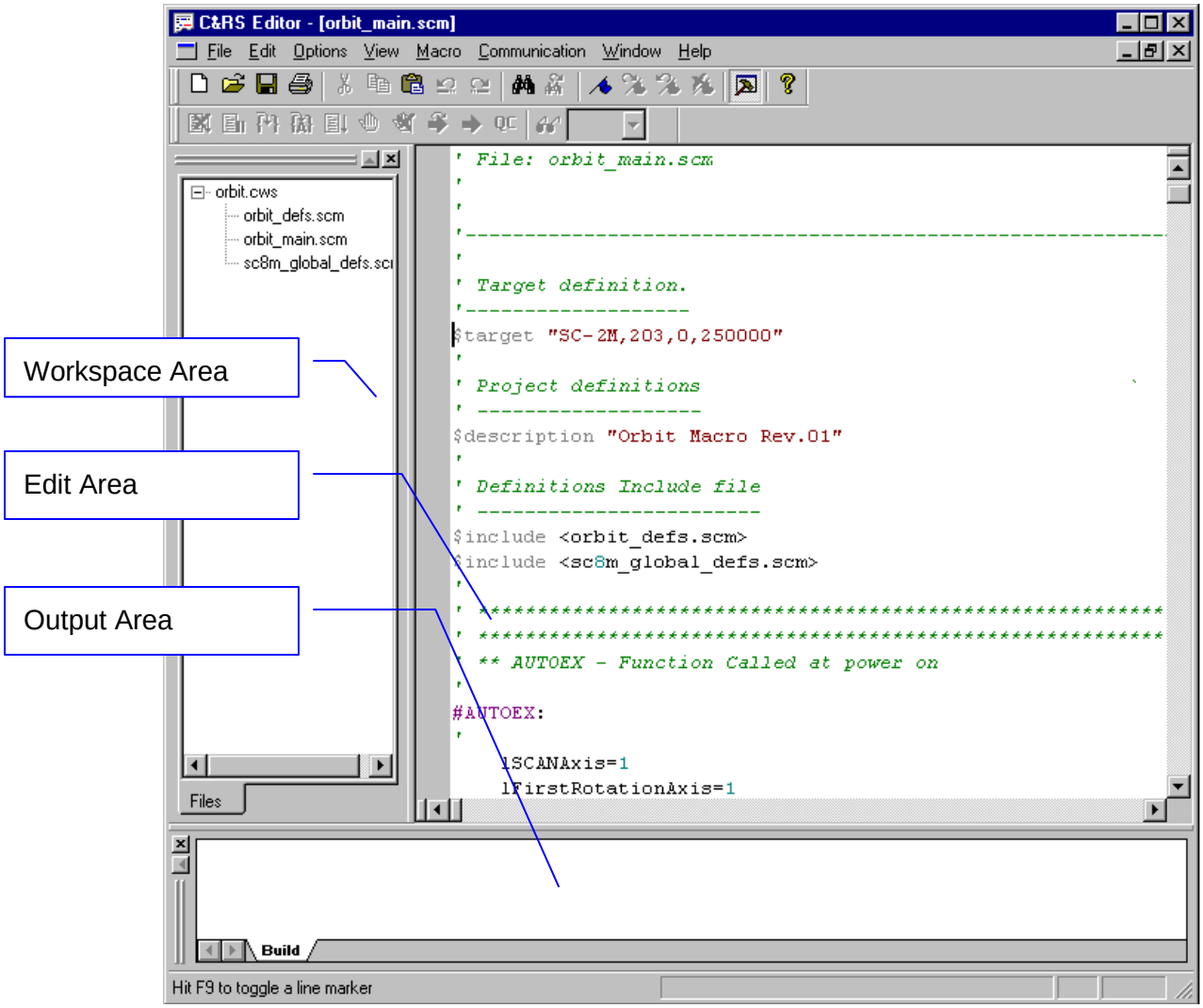


Figure 57: Macro Editing

Edit mode enables to edit macro files.

In this mode the editor gives a powerful text editor with syntax coloring, see Figure 57.

The Macro File Edit components in Edit Mode:

- Workspace Area – see section 19.3, Part IV.
- Edit Area – this area includes the opened files for editing. Files with ".scm" extension are syntax-colored, other files are in black and white colors.
- Output area – includes macro compilation errors and results.

Syntax coloring is divided into word groups:

- Default text.
- Numbers.
- Symbols.
- Strings.
- Comments.
- Directive Commands (such as $define, $target).
- Labels (such as #AUTOEX, #HOME_X).
- High Level Word (such as if, else, for, while).

For more information on colors settings see 19.7.3.2, Part IV.

19.5 Macro Downloading

By downloading macro to the controller, the user can find two types of errors:

- Syntax errors – errors in syntax such as "for" loops (paring errors).
- Program flow errors – errors detected by the debugging process.

Downloading Macro step-by-step:

1. Open the macro file in the Edit Area. If working with workspace select the main macro file.
2. Verify that the relevant controller shell is opened and communicating with the controller.
3. On the "Macro" menu click "Download Macro".
4. The result of the download appears in a message: if the download succeeds than an OK message appears; otherwise, the errors are listed in the Output Area.

If the macro has errors, they appear in the Output area. The syntax of the download errors is: <file name> (<line number>), <Error description> :

- File name – the file which contains errors. Note: a macro can include more than one file.
- Line number – the line number points out on the error location.
- Error Description – a short description of the error.

19.6 Macro Debugging

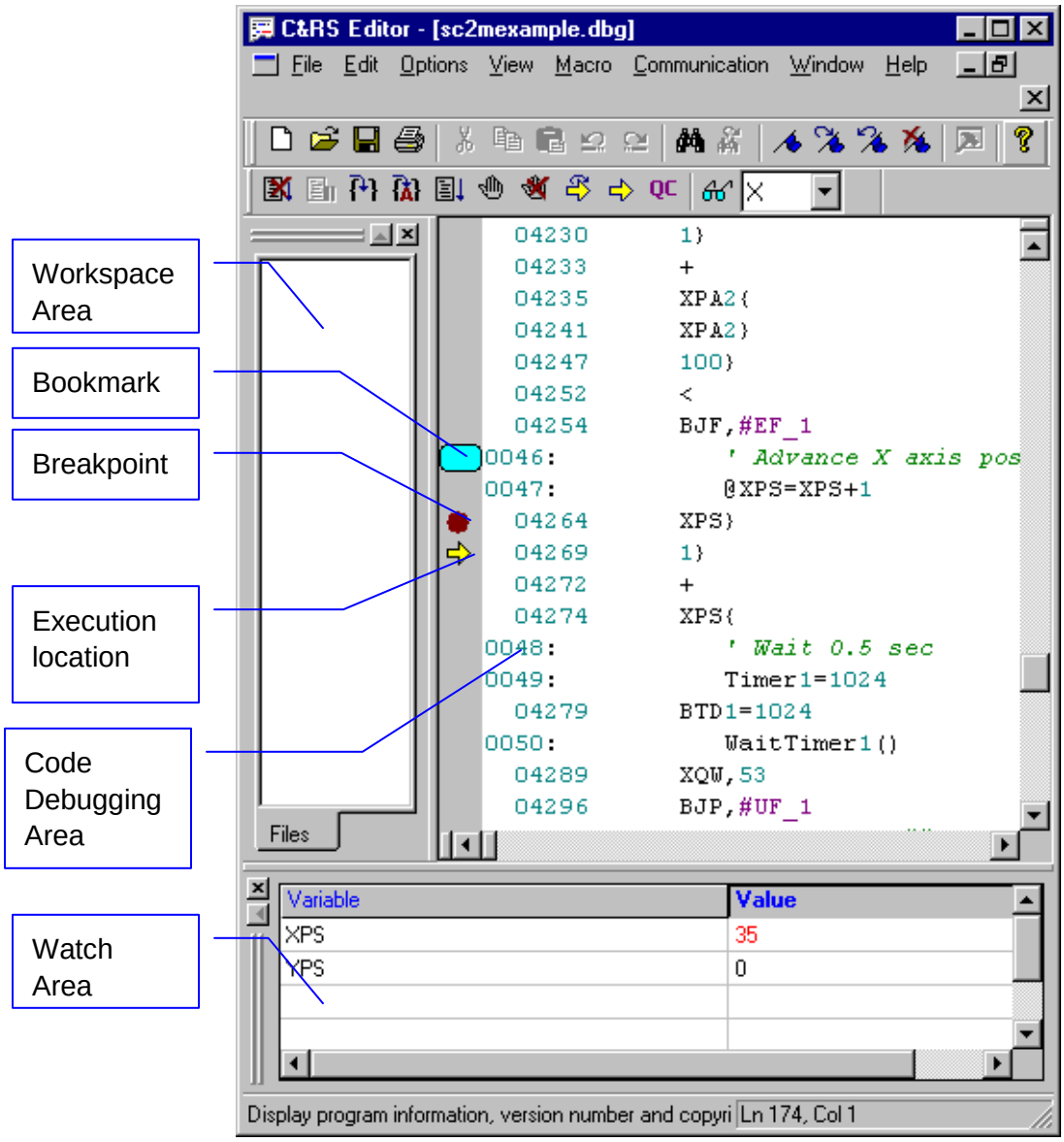


Figure 58: Macro Debugging

The macro Debug Mode enables debugging a macro that currently resides in the controller (see Figure 58).

To switch the Macro File Editor Application to the macro Debug Mode, follow the next instructions:

- If the macro is not in controller, first download the macro with "Download Macro" menu item or "Save and Download Current Macro" menu item (see sections 19.7.5.1 and 19.7.5.2, Part IV).
- If the macro is already in the controller, on the "Macro" menu click on one of the following:
  - If macro is not opened in the editor, click "Debug Macro" (see section 19.7.5.3, Part IV).
  - If macro is opened in the editor, click "Debug Current Macro" (see section 19.7.5.4, Part IV).

After the Macro File Editor Application is in debug mode, the editor shows the following components, see Figure 59:

- Code Debugging Area – shows the current debugging status: macro commands, their lines and low-level commands' translation.
- Line Number – editor macro line number.
- Controller Memory Address – controller's commands memory address.
- User Macro Command – pre-compiled macro command, as appears in the macros ".scm" file.
- Macro Command Interpretation – interpretation of the macro command in low-level controller commands.

As seen in Figure 59, the debugging area shows the user macro command and the corresponding low-level commands. The debugger runs on the low-level commands and in each line resides one controller command.

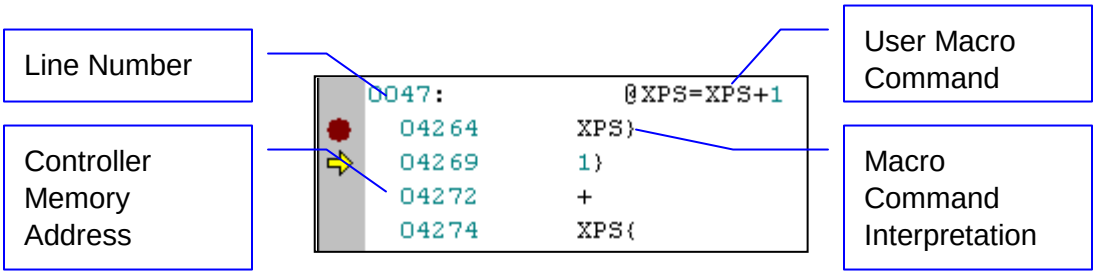


Figure 59: Debugging Area Example

- Debug Toolbar – the debug toolbar is enabled for all the debug toolbar options see section 19.8.2, Part IV.
- Watch Area – this area shows watch variables and their values. This area enables:
  - Adding Variables to the Watch Area:
    - ♦ Add directly to the Watch Area by editing the variable.
    - ♦ Select a variable from the Code Debug Area, left-click on "Add to Watch Menu".
  - Remove watch variable by selecting the variable from the variable list and press "Delete" button.
  - To edit a watch variable, double click the variable. To apply changes press "Enter".
  - Updating the watch variable list (see section 19.7.5.17, Part IV). A Variable value that changed from the last update is colored with Red, if the value did not change its color is black.
- Source Icons Area – this area shows breakpoints, bookmarks and the current execution point.
- Bookmarks – blue cubes in the Source Icons area.
- Breakpoints – red circles in the Source Icons area.
- Execution Location – yellow arrow in the Source Icons area marks the current execution location.

19.7 Menus

19.7.1 File Menu

The "File Menu" section covers all File Menu items.

19.7.1.1 New

The "New" menu item creates a new empty macro file.

19.7.1.2 Open

The "Open" menu item opens a macro file to edit.

19.7.1.3 Close

The "Close" menu item closes the top most opened macro file.

19.7.1.4 Workspace

The "Workspace" menu item shows the workspace options.

19.7.1.4.1 New Workspace

The "New Workspace" menu item creates an empty new workspace. By choosing this menu item the user can select or enter a workspace name, under which the workspace file is saved.

19.7.1.4.2 Open

The "Open" menu item opens a workspace from a file. By choosing this menu item the user can select or enter the workspace file.

19.7.1.4.3 Save

The "Save" menu item saves the opened workspace.

19.7.1.4.4 Save As

The "Save As" menu item saves the opened workspace to a user defined name.

19.7.1.4.5 Close

The "Close" menu item closes the opened workspace.

19.7.1.4.6 Add File

The "Add File" menu item adds file to a workspace. By choosing this menu item the user can select a file to be added.

19.7.1.4.7 Remove File

The "Remove File" menu item removes the highlighted file from workspace.

19.7.1.5 Save

The "Save" menu item saves the top most opened file.

19.7.1.6 Save As

The "Save As" menu item saves the top most opened file to a different name.

19.7.1.7 Save All

The "Save All" menu item saves all files opened in the Edit Area.

19.7.1.8 Print

The "Print" menu item prints the top most opened file.

19.7.1.9 Print Preview

The "Print Preview" menu item previews the file to print before printing.

19.7.1.10 Print Setup

Print Setup enables the selection of printer and its menu items before printing.

19.7.1.11 File Locations

The "File Location" menu item prompts the File Location dialog box (see Figure 60).

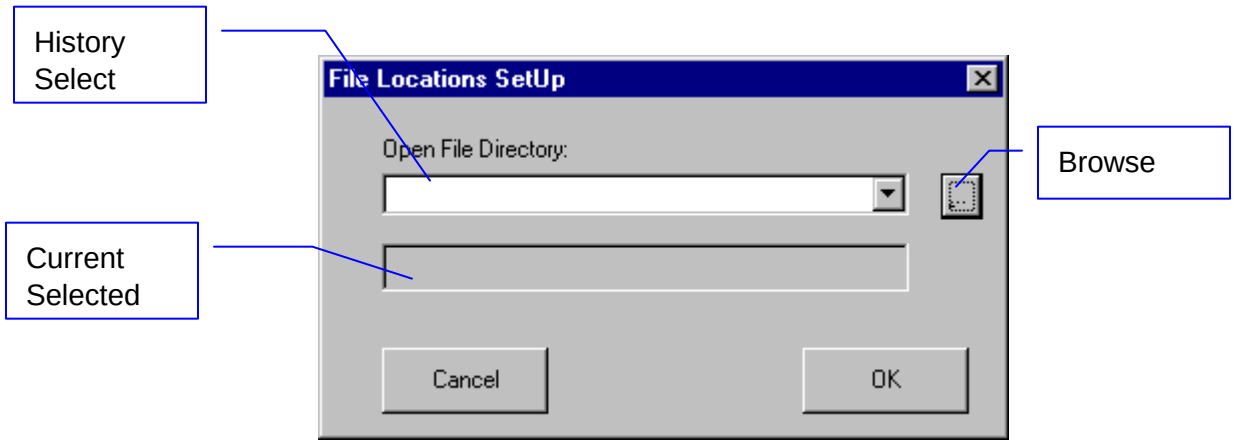


Figure 60: Source Code Edit File Location Dialog

This dialog box enables editing the open files directory location. This directory is the directory to open in the file browser.

Every time the user chooses "Open", "Save" or "Download Macro", the editor opens the file browser at the location specified in this dialog box.

The dialog components are as follows:

- Browse button - enabling browsing to the desired location.
- History Select – enables to select a location previously selected, from a combo box.
- Current Selected – shows the currently configured location.
- OK button – closes the dialog and applies changes.
- Cancel button – closes the dialog and discards changes.

19.7.1.12 Recent Files

The "Recent Files" menu item shows a list of recently opened files. The user can choose a recently opened file to open from the list.

19.7.1.13 Recent Workspace

The "Recent Workspace" menu item shows a list of recently opened workspaces. The user can choose a recently opened workspace to open from the list.

19.7.1.14 Exit

The "Exit" menu item closes the Macro File Editor Application.

19.7.2 Edit Menu

The "Edit Menu" section covers all Edit Menu items.

19.7.2.1 Undo

The "Undo" menu item enables the user to undo the last operation.

19.7.2.2 Redo

The "Redo" menu item enables the user to redo the last "undo" operation.

19.7.2.3 Cut

The "Cut" menu item cuts (remove + copy) the highlighted text.

19.7.2.4 Copy

The "Copy" menu item copies the highlighted text.

19.7.2.5 Paste

The "Paste" menu item pastes the text retrieved by the "Cut" or the "Copy" command.

19.7.2.6 Select All

The "Select All" menu item selects all the text in the top most opened file.

19.7.2.7 Find

The "Find" menu item searches for a string in the top most open file.

Selecting this menu item shows the "Find" dialog box, see Figure 61.

Figure 61: Find Dialog Box

19.7.2.8 Replace

The "Replace" menu item replaces the given text with another given text.

Selecting this menu item shows the "Replace" dialog box, see Figure 62.

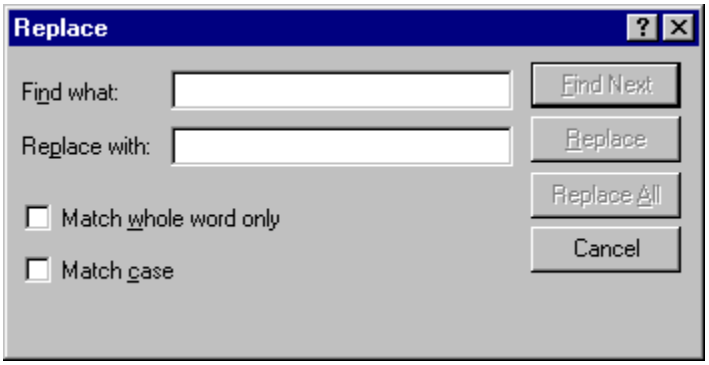


Figure 62: Replace Dialog Box

19.7.2.9 Go To

The "Go To" menu item jumps to a given line number by the user.

19.7.3 Options Menu

The "Options Menu" section covers all Options Menu items.

19.7.3.1 Editor

The "Editor" menu item enables the user to change the editor options. This menu item shows the "Editor Options" dialog box, see Figure 63.

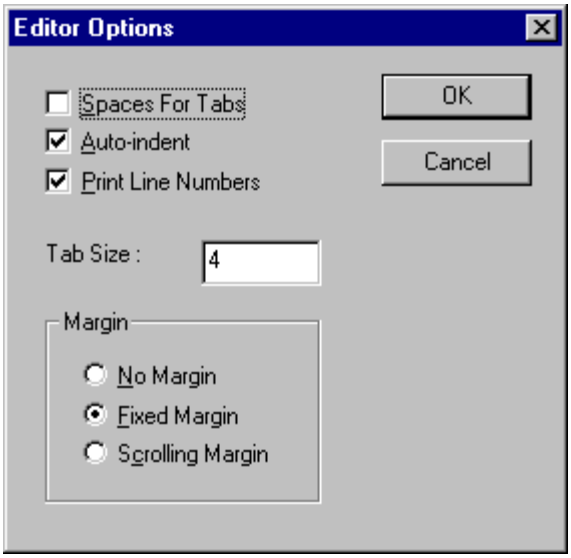


Figure 63: Editor Options Dialog Box

Editor Options dialog box enables:

- Spaces for Tabs – changes tab chars with space bar chars.
- Auto Indent – enables/disables automatic indentation.
- Print Line Number – enables/disables printing line number.
- Tab Size – number of space bar chars per tab char.
- The "Margin" menu item is not supported in the current FlexDC software.

19.7.3.2 Colors

By choosing the "Colors" menu item the user can change the editor text colors. This menu item shows the "Colors" dialog box, see Figure 64.

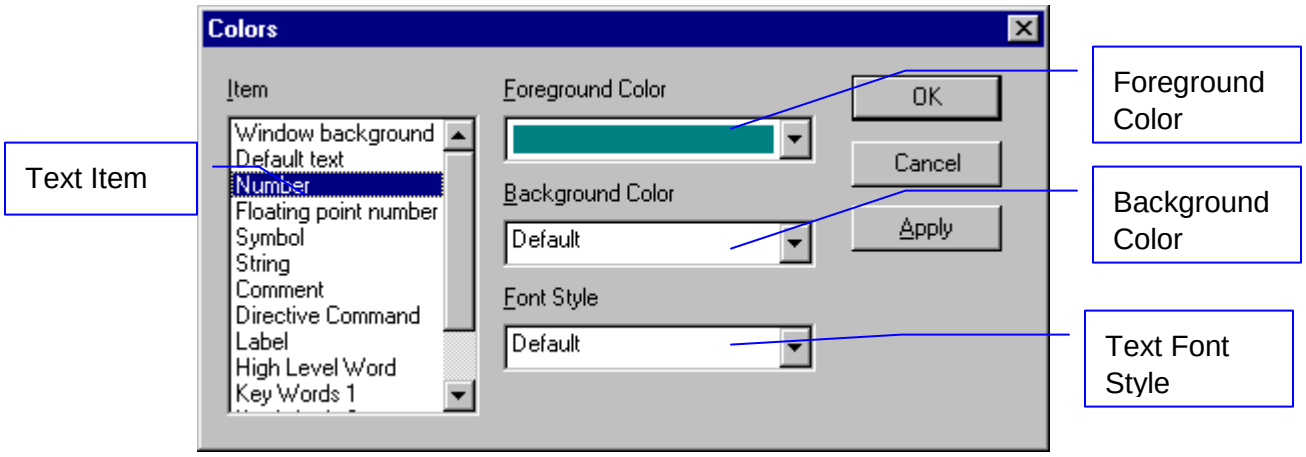


Figure 64: Colors Dialog

Colors dialog components:

- Text Item – selects the text type to be changed.
- Foreground Color – text foreground color.
- Background Color – text background color.
- Text Font Style – Normal, Bold or Italic.

Number text color and font setting step-by-step:

1. Select the "Text Item" to edit.
2. Select the text foreground.
3. Select the text background. "Default" is the page color.
4. Select the text font style.
5. To apply changes for this document press "Apply" and than press "OK".

19.7.4 View Menu

The "View Menu" section covers all View Menu items.

19.7.4.1 Main Toolbar

The "Main Toolbar" menu item shows / hides the main toolbar.

19.7.4.2 Debug Toolbar

The "Debug Toolbar" menu item shows / hides the debug toolbar.

19.7.4.3 Status Bar

The "Status Bar" menu item shows / hides the Status Bar, at the bottom of the editor screen.

19.7.4.4 Output

The "Output" menu item shows / hides the Output area (see Figure 57).

19.7.4.5 Watch

The "Watch" menu item shows / hides the Watch area, only in debug mode (see Figure 58).

19.7.4.6 Work Space

The "Work Space" menu item shows / hides the Work Space area (see Figure 57).

19.7.5 Macro Menu

The "Macro Menu" section covers all Macro Menu items.

19.7.5.1 Download Macro

The "Download Macro" menu item enables to download macro file to the controller. This menu item is available only when one the Nanomotion Shell Application is open and connected to controller.

19.7.5.2 Save and Download Macro

The "Save and Download Macro" menu item saves the current edited macro and downloads it. This menu item is available only when one the Nanomotion Shell Application is open and connected to controller.

19.7.5.3 Debug Macro

The "Debug Macro" menu item enables to debug a given macro file.

This menu item is available only when one the Nanomotion Shell Application is open and connected to controller. The given macro file must be the same as the macro that resides in the controller.

19.7.5.4 Debug Current Macro

The "Debug Macro" menu item enables to debug the current edited macro. This menu item is available only when one the Nanomotion Shell Application is open and connected to controller. The edited macro must be the same as the macro that resides in the controller.

19.7.5.5 Debug Macro X

The "Debug Macro X" menu item switches the debugging to macro X (Note: there are 2 possible macros in the controller X, Y).

19.7.5.6 Debug Macro Y

The "Debug Macro Y" menu item switch the debugging to macro Y. (Note: there are 2 possible macros in the controller X, Y).

19.7.5.7 Reset Program

The "Reset Program" menu item resets the current selected macro execution point.

19.7.5.8 Break Macro Program Execution

The "Break Macro Program Execution" menu item halts current selected macro running. This menu item is enabled only when a macro is running.

19.7.5.9 Trace One Step

The "Trace One Step" menu item executes one command in the current selected macro.

19.7.5.10 Animate Macro Execution

The "Animate Macro Execution" menu item animates the current selected macro execution by running the macro step after step with a delay between the steps. The animation appears on the screen followed by a current executed line yellow arrow.

19.7.5.11 Go from Current Pointer Location

The "Go from Current Pointer Location" menu item runs a current selected macro from a current location.

19.7.5.12 Insert/Remove Breakpoint

The "Insert/Remove Breakpoint" menu item adds or removes breakpoints.

To add breakpoints follow the next steps:

1. Move the cursor to the desired line.

2. Press F9 button or select this menu item.

To remove breakpoints follow the next steps:

1. Move the cursor to the breakpoint line.

2. Press F9 button or select this menu item.

19.7.5.13 Remove All Breakpoints

The "Remove All Breakpoints" menu item removes all break points from a current selected macro.

19.7.5.14 Set Next Statement

The "Set Next Statement" menu item sets the next line to be executed.

To Set Next Statement execution line follow the next steps:

1. Move the cursor to the desired line.

2. Select this menu item.

19.7.5.15 Show Next Statement

The "Show Next Statement" menu item scrolls the edit area to the next line to be executed. Use this menu item to find the current macro position.

19.7.5.16 Show Run Time Error

The "Show Run Time Error" menu item shows the current selected macro Run Time Error (QC).

19.7.5.17 Update Watch List

The "Update Watch List" menu item updates the Watch Area variables.

19.7.6 Communication Menu

The 'Communication Menu" section covers all Communication Menu items.

19.7.6.1 Enable Shell Com

The "Enable Shell Com" menu item is used to reestablish connection with Nanomotion Shell Application, if a connection was lost.

19.7.7 Window Menu

The "Window Menu" section covers all Window Menu items.

19.7.7.1 New Window

The "New Window" menu item opens the top most opened file again in a new window.

19.7.7.2 Cascade

The "Cascade" menu item cascades all the opened files in the Edit Area.

19.7.7.3 Tile

The "Tile" menu item tiles all the opened files in the Edit Area.

19.8 Toolbars

The Toolbars" section gives a description of the application's toolbars.

19.8.1 Edit Toolbar

Figure 65: Edit Toolbar

The "Edit Toolbar" (see Figure 65) enables the following menu items:

-  this toolbar section is dedicated for file manipulation such as "New", "Save", "Open" and "Print" (see section 19.7, Part IV).
-  this toolbar section is dedicated for editing the text, such as "Cut", "Copy", "Past", "Undo" and "Redo" (see section 19.7, Part IV).
-  this toolbar section is dedicated for text searching in the current opened file (see section 19.7, Part IV).
-  this toolbar section is dedicated for bookmarks. It enables to "Add", "Go to Next", "Go to Previous" and "Remove all Bookmarks".
-  this toolbar button shows/hides the Output Area.
-  this toolbar button is for showing the "About" dialog box.

19.8.2 Debug Toolbar

Figure 66: Debug Toolbar

Debug Toolbar (see Figure 66) enables the following menu items:

-  this toolbar button is for halting the debugging and resetting the current macro (see section 19.7.5.7, Part IV).
-  this toolbar button is for breaking the current, selected macro execution (see section 19.7.5.8, Part IV).
-  this toolbar section is dedicated for tracing one step (see section 19.7.5.9) and macro animation (see section 19.7.5.10, Part IV).
-  this toolbar button is for running the current, selected macro (see section 19.7.5.11, Part IV).
-  this toolbar section is dedicated for breakpoints. It enables adding, removing (see section 19.7.5.12) and removing all break points (see section 19.7.5.13, Part IV).
-  this toolbar section is dedicated for setting and showing the next macro statement (see section 19.7.5.14).
-  this toolbar button is for showing the current, selected macro Run Time Error (see section 19.7.5.16, Part IV).
-  this toolbar button refreshes the Watch List and updates the Watch Area (see section 19.7.5.17, Part IV).
-  this toolbar combo-box is for switching between macros. The combo-box selection changes according to the Nanomotion Shell Application and editor, by communicating with FlexDC. The combo-box includes 2 possible macros.

19.9 Appendix B – Macro File Editor Application Keyboard Shortcuts

Key	Purpose
Ctrl+D	Download macro
F7	Save and download current macro
Alt+G	Debug macro
Ctrl+F7	Debug current macro
Ctrl+R	Reset program
Ctrl+B	Break macro execution
F10	Trace one step
Ctrl+E	Animate macro execution
F5	Run program from current pointer location
F9	Insert/Remove breakpoint
Ctrl+T	Show next statement
Ctrl+Q	Show Run Time Error
Ctrl+N	New file
Ctrl+O	Open file
Ctrl+S	Save file
Ctrl+P	Print file

Table 48: Macro File Editor Application Keyboard Shortcuts

Part V – SCServer COM/DCOM Interface Library

20 Introduction

Part V describes the COM software interface module for the Nanomotion FlexDC servo controller. The purpose of this module is to provide a simple generic server interface for the FlexDC servo controller.

The defined upper level implementation method for the required interface is a Microsoft © COM-DCOM based server module.

The underlying communication interfaces with the actual FlexDC hardware and firmware are based on the following existing hardware communication links:

- **RS232**: based on the standard supported host PC serial communication links.
- **CAN Bus**: based on a KVASER CAN board.

Note:

- *Future versions of the DCOM server interface may support additional links (such as USB for example, etc).*

The server supports simultaneous low-level interfaces, thus supporting, for example, three controllers: one working with CAN Bus, another one with RS232 COM1, and an additional one with RS232 COM2.

Important note:

- *The interface is designed and implemented as a single threaded interface. This means that at any given time only one controller can be accessed. This of course does not limit the number of processes that can access the server interface. All access requests are queued and handled one at a time by the server (standard single thread Server interface with SINGLETON interface).*

20.1 Interface Specifications

This section shortly describes the specifications of the interface.

- The DCOM server provides interfaces to all of the FlexDC controller parameters. Each FlexDC controller variable is defined in the DCOM server as a server object parameter or method.
- It is possible to communicate with any controller via RS232 or via CAN. This, of course, is limited to the number of CAN ports (0-2047), and COM ports (computer dependant). Note that each controller requires at least two (unique) addresses, one for transmit and one for receive. Note that actual number of nodes supported (number of devices) may be limited by the hardware to a much lower value. Please consult Nanomotion experts for further information regarding this issue.
- The DCOM server may run without any hardware (cable, card or servo controller, etc.). This may be done for all hardware connections and/or for individual ones as well. Note that in the case of this *debug mode,* (refer to the OpenDevice/Ex methods) all calls to the DCOM server will result with *"**0**"* as the reply.
- Once an error occurs, the COM Object throws an error. This error can be caught with the try-catch pair in VC or with the *On Error* in VB. The user can then read the *LastError* parameter thus returning the relevant error code, e.g. Timeout, WriteError, CanAddrError, ReadError etc, or, alternatively read the description returned by the com_error object. These codes are defined values.
- The following DCOM method was defined in order to allow multiple devices interfaces (through all supported underlying communication links, currently RS232 and CAN):

  OpenDevice(short sDevID,
  short sControllerType,
  short sCOMMType,
  short sAddrRx,
  short sAddrTx,
  long lSerialBaud,
  short sSerialPort,
  short sDebugMode)

  Whereas…

- ♦ sDevID - Device ID number to be used by all sub-sequent methods/parameters interfacing this device. Note that this number is a global ID number that should be used by all processes accessing the device interface.
- ♦ sControllerType – currently, either MCD or SC family controllers.
- ♦ sCOMMType – currently CAN bus or serial.
- ♦ sAddRx,sAddTx – CAN receive and transmit addresses.
- ♦ lSerialBaud – RS232 serial baud rate.
- ♦ lSerialport – RS232 serial port.
- ♦ DbgMode – flag whether this device works in debug mode or not.

- The function fails if the DevID is already open (on a different configuration than stated) or if serial COM port is busy etc.
- Once this method was called with identical configuration as another client (ID and communication configuration), the COM object increments an internal variable stating the number of clients connected to this DevID. This internal variable is decremented once the CloseDevice method is called. One the variable = 0, the device is really freed.
- In order to enquire whether a configuration is connected, use the IsConnected() parameter, which returns the DevID of the connected configuration. (or –1 if not connected).
- In order to enquire the connection configuration of a DevID, call the GetDevStatus method which returns the full connection configuration. (As opposed to the IsConnected parameter which returns a DevID).
- The DevID is used to interface all DCOM methods (Return errors if was not opened, i.e. SC.Speed(Dev1,Axis1,5000) where Dev1 is not defined). A method to inquire existing DevID properties will also be defined.
- In addition, a CloseDevice() function was of course implemented to allow closing a previously opened connection. This is to allow multiple connections of the same device, which may be required for example for debugging sessions.

20.2 The SCServerScope Application

The DCOM interface uses Registry variables to define the type of CAN board used and also for the CAN baud rate.

A simple dedicated Windows application and code is provided to access the following global Registry Database parameters (see Figure 67):

- CAN Card Type.
- CAN Baud rate.
- CAN Buffer Size.
- RS232 Buffer Size.

These variables are an integral part of the COM interface.

The application is known as SCServerScope Application (or the SCServerScope.exe). This application includes in its file menu, under “COM Registry”, the "Registry Setter" dialog box.

This dialog box enables the user, as mentioned to set and read the current CAN card type, baud rate, buffer size and RS232 buffer size we are using.

The Registry path is as follows:

HKEY_LOCAL_MACHINE\SOFTWARE\CRS\SCServer

20.3 Keys Used by the Application

CANCardBaud – 0 stating 1000 Baud.

1 stating 500 Baud.

2 stating 250 Baud.

3 stating 125 Baud.

4 stating 100 Baud.

5 stating 50 Baud.

CANCardType - 0 stating the PIC – ISA card.

1 stating the Kvaser – PCI card.

2 stating the ESD – PCI card.

3 stating the CRS – ISA card.

CANBufferSize - Supports 1k - 16k buffer size.

RS232BufferLength – Supports 65k – 1M buffer sizes.

20.4 Product Notations Revisions

The DCOM server object modules are identified under the following notations:

Object Name: SCServerExe, Revision 2.31

Object Interface Name: ISCServerExeInterface.

21 Getting Started

21.1 Setting up the Object, Baud and Card Type

1. The DCOM Server setup files are intalled together with the Nanomotion Shell Application installation. After the installation process is complete the DCOM server is ready to use. The DCOM module uses the Windows Registry database for some initial settings as defined below:

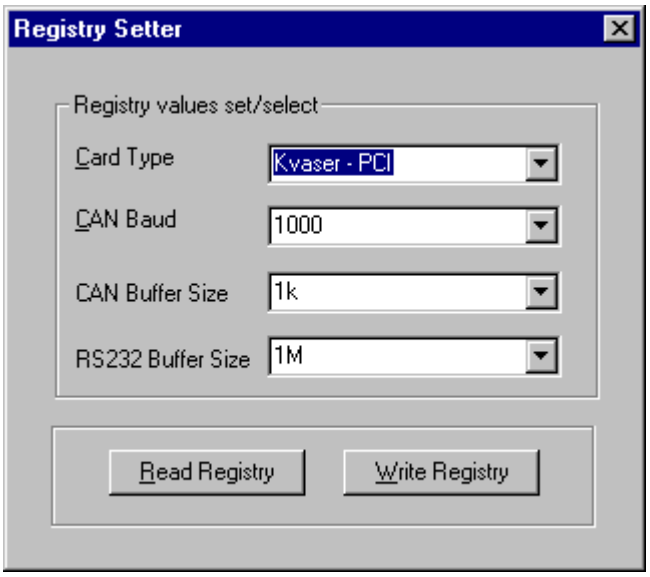


Figure 67: DCOM Communication Registry Setup

2. Double click the "SCServerScope" on your desktop.
3. On the "File" menu, click "COM Registry", in the SCServerScope – Nanomotion Ltd. dialog box, to set the registry variables, see Figure 67.

21.2 Using the Object from VB

1. Open VB and open a new project (Standard exe project.)
2. Open the references dialog box, see Figure 68. (Menu – Project→ References).

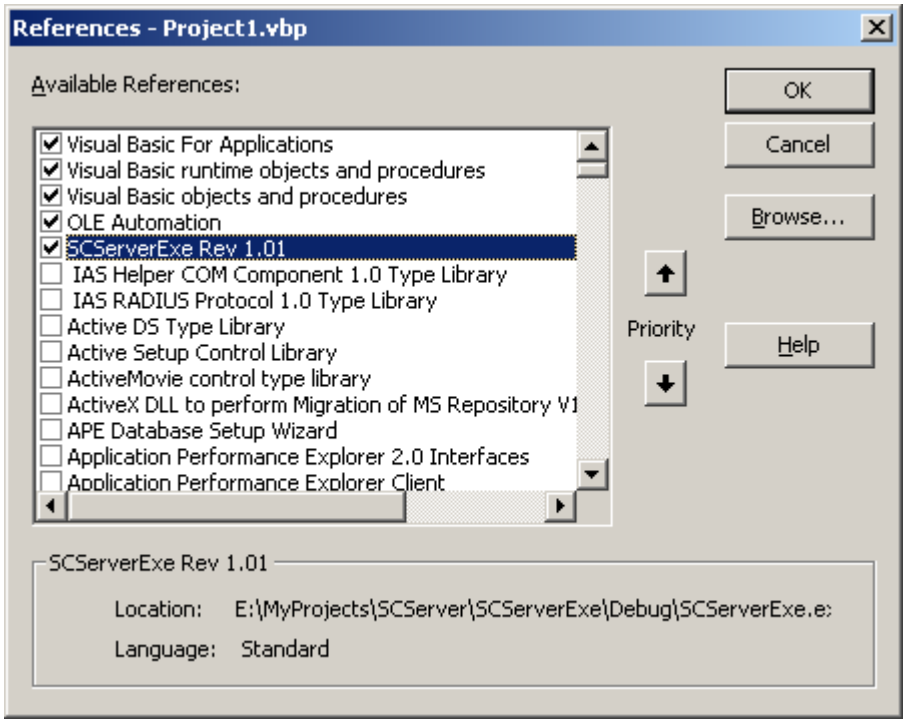


Figure 68: References Dialog Box

3. Select the SCServerExe option.
4. From the declarations area define a interface to the object e.g. *Dim SC As New SCServerExeInterface.*
5. All parameters and methods may be accessed by using the defined interface parameter. (In the example above, use SC as the parameter name.)

21.3 VB Code Example

```
' Declarations:
Dim SC As New SCServerExeInterface

Const RS232_COMM = 21
Const CAN_COMM = 20
Const X_AXIS = 88
Const Y_AXIS = 89

' Form Load
Private Sub Form_Load()

Dim ddd As Long
Dim str As String
Dim aa As Integer
Dim bb As Integer
Dim cc As Integer
Dim dd As Integer
Dim ee As Integer
Dim ff As Integer
Dim gg As Integer
Dim hh As Integer
Dim ll As Long

On Error GoTo MyError:

SC.GetDeviceStatus 0, aa, bb, cc, dd, ee, ll, gg, hh
If aa <> 1 Then
```

```
        ' Open CAN Device
SC.OpenDevice 0, 50, CAN_COMM, 600, 500, 0, 1, 0
End If

        ' Read SC Version
str = SC.FirmwareVersion(0, X_AXIS)



Exit Sub

MyError:
str = Err.Description
MsgBox str, vbOKOnly, "Error"

End Sub
```

21.4 Using the Object from Visual C

1. Import the SCServerExe type library into the project by writing the following line:

```
#import  "SCServerExe.tlb" no_namespace
```

2. Declare a variable of type ISCServerExeInterface , e.g.:

```
ISCServerExeInterface *piSC
```

3. Open the COM Object using the regular COM methods.
4. VC code Example:

```
#import  "SCServerExe.tlb" no_namespace

ISCServerExeInterface *piSC  = NULL;


    try
      {
            HRESULT hr = CoCreateInstance (__uuidof (SCServerExeInterface)
            ,NULL,CLSCTX_ALL,_uuidof(ISCServerExeInterface),reinterpret_cast <void**>
            (&piSC) )

            piSC->OpenDevice(0, CRS_SC_2M, CAN_COMMUNICATION ,600, 500, 0, 1,
            0) ;
        lVal  = piSC->GetPosition(0,'X') ;
      }
      catch(_com_error e)
      {
        char str[150] ;

        _bstr_t desc = e.Description();
        if (desc.length() > 0)
        {
            sprintf(str,"Description: %s\n", (LPTSTR)desc);
            MessageBox(str," COM-Error",MB_OK) ;
        }
      }
      CoUninitialize();
      if (piSC)
      {
        piSC->Release() ;
}
```

22 Object Parameters and Methods Syntax

22.1 Object Parameters

In Table 49, a short description of all the existing interface commands implemented as object parameters is given, along with the following additional information: Keyword is axis related or not, and parameter if is read only.

Accessing the FlexDC parameters have the identical syntax, with the exception of a few parameters (are documented in the FlexDC COM interface keyword):

Param(short sDEVID,short sAxis) – whereas the *sDEVID* is the opened device, and the *sAxis* is the letter of the required axis.

FlexDC COM Interface Keyword	FlexDC			Parameter Description
	Key word	Axis Relate	Read Only	
AbsolutePosition	AP	Yes	No	Absolute position
Acceleration	AC	Yes	No	Acceleration
AnalogInput	AI	No	Yes	Analog Input
AnalogInputDeadBand	AD	Yes	No	Analog Dead-Zone
AnalogInputGain	AG	Yes	No	Analog Gain
AnalogInputGainFactor	AF	Yes	No	AnalogInputGainFactor
AnalogInputOffset	AS	Yes	No	Analog Offset
AnalogInputsFilterCoefficient	XF	Yes	Yes	Analog Inputs Filter Coefficient
AnalogOutput	AO	Yes	No	Analog Output
AnalogOutputsScale	OS	Yes	No	The scaling of the Analog Output

FlexDC COM Interface Keyword	FlexDC			Parameter Description
	Key word	Axis Relate	Read Only	
ArrayElement(short sDEVID short sAxis,short sArrayType,short sIndex) **Description:** **sArrayType – AR=0 ,CV=1, IA=2,TD=3,QB=4,PA=5,ZI=6,XI=7, QF=8,CA=9,DA=10,PG=11,GP[14]= 12,FA=13,FV=14,FF=15,KD=16,KI =17,KP=18,RG=19,ZE=20,ET=21** **sIndex = Index in array.**	AR CV IA TD QB PA ZI XI QF CA DA PG GP FA FV FF KD KI KP RG ZE ET	Yes	No	Assigns and reads array element variables. e.g. PA,AR etc.
CANReceiveAddress	RA	No	No	The SC CAN Rx Address
CANTransmitAddress	TA	No	No	The SC CAN Tx Address
CANBaudRate	CB	No	No	CAN Baud Rate.
CommErrorCode	EC	No	No	Communication Error Code
Configuration	CG	Yes	No	Configuration
ContinuousCurrentLimit	CL	Yes	No	Torque Limit
CurrentFilterFactor	CF	Yes	No	The Current Filter Factor
CompareDistance	CD	Yes	No	Compare Distance

[14] This command is obsolete in current FlexDC firmware.

FlexDC COM Interface Keyword	FlexDC			Parameter Description
	Key word	Axis Relate	Read Only	
CompareMode	CM	Yes	No	Compare Mode
DACCommand	TC	Yes	No	DACCommand
DACLimit	TL	Yes	No	DAC Limit
DACOffset	DO	Yes	No	DAC Offset
Deceleration	DC	Yes	No	Deceleration
DesiredPosition	DP	Yes	Yes	Report current Servo Position command
ECAMEndIndex	EE	Yes	No	The last index in ECAM mode
ECAMGapSize	EG	Yes	No	The gap size in ECAM mode
ECAMInterpolationMode	EI	Yes	No	The interpolation mode in ECAM mode
ECAMStartIndex	ES	Yes	No	The first index in ECAM mode
ECAMWrapIndex	EW	Yes	No	The wrap index in ECAM mode
EndMotionReason	EM	Yes	Yes	The last motion Reason for defined Axis
FirmwareVersion	VR	No	Yes	SC Firmware Version
FollowerGearingRatio	FR	Yes	No	Follower Gearing ratio
HighLimit	HL	Yes	No	Position High limit
InputsLogic	IL	No	No	Digital input logic
InputsPort	IP	No	Yes	Digital Input Port
LowLimit	LL	Yes	No	Position Low limit
MaximumError	ER	Yes	No	Error limit
MotionMode	MM	Yes	No	Motion Mode
MotionStatus	MS	Yes	Yes	Report Motion status
MotorCurrent	IM	Yes	Yes	The Current motor current
MotorFault	MF	Yes	Yes	The Last Motor Fault
MotorOn	MO	Yes	NO	Motor On.
NoControl	NC	Yes	No	No Control

FlexDC COM Interface Keyword	FlexDC			Parameter Description
	Key word	Axis Relate	Read Only	
ObjectVersion **Description:** **The *get_ ObjectVersion* returns short* pVal. To version #. (refer idl file)**	N/A	No	Yes	The SCServerExe COM Object's version.
OutputsLogic	OL	No	No	Digital out logic
OutputsMode	OM	Yes	No	Special Outputs Mode
OutputsPort	OP	No	Yes	Digital Output Port
OutputBit	OS OC	No	No	Set/Get OutputBit
PeakCurrentLimit	PL	Yes	No	Peak Current Limit
PeakCurrentDuration	PD	Yes	No	Peak Current Duration
PIDDifferentialTerm	KD	Yes	No	Digital Filter Zero
PIDGain	GA	Yes	No	Digital filter Gain
PIDGainTerm	KP	Yes	No	Digital filter Gain
PIDIntegralTerm	KI	Yes	No	Digital filter Integration constant
PIDOutput	PO	Yes	Yes	PID Output
PIDFeedForwared	FF	Yes	No	PID Feed Forwared
Position	PS	Yes	No	Current Position
PositionError	PE	Yes	Yes	Position Error
ProfileSmoothing	WW	Yes	No	Profile Smoothing
Program Status	QR	Yes	Yes	Current Program Status
ProgramPointer	QP	Yes	No	Current Program Pointer
ProgramRTE	QC	Yes	Yes	Last Program RTE
RecordingGap	RG	No	No	Time Interval
RecordingLength	RL	No	No	Recording Array Upper limit
RecordingStatus	RR	No	Yes	Recording Function Enable
RecordingVariables	RV	Yes	No	Data collection mask
RecordingTriggerValue	TV	Yes	No	Recording Trigger Value
RecordingTriggerSource	TS	Yes	No	Recording Trigger Source
RecordingTriggerPosition	TP	Yes	No	Recording Trigger Position

FlexDC COM Interface Keyword	FlexDC			Parameter Description
	Key word	Axis Relate	Read Only	
RelativePosition	RP	Yes	No	Relative Position
SearchSpeed	SS	Yes	No	Search Speed
SpecialMode	SM	Yes	No	Special Mode e.g. repetitive motion
Speed	SP	Yes	No	Linear velocity
StatusRegister	SR	Yes	Yes	Status Register
TargetRadius	TR	Yes	No	Target Radius
TargetRadiusTime	TT	Yes	No	Target Radius period
Velocity	VL	Yes	Yes	Actual Velocity
WaitPeriod	WT	Yes	No	Wait Time
LastError	--	No	Yes	Internal COM parameter returning the last error in the system
UserMode	UM	No	No	MCD only

Table 49: Interface Commands

22.2 Object Methods

In Table 50, a short description is given to all existing interface commands methods.

Method (Keyword)	Function Parameters	FlexDC Keyword	Function Description
Abort	short sDEVID – Opened Device ID short sAxis – Defined Axis for operation.	**AB**	Abort Motion on defined axis
Begin	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation.	**BG**	Begin Motion on defined axis
CloseDevice	short sDEVID– Opened Device ID	**N/A**	Closes an open device
GetDeviceStatus/Ex	short sDevID– Opened Device ID short *iDevOpen – Flag if deice is open or not. short *iControllerType – Controller type the object is connected to short *iCOMMType – Communication type the object is connected to short *iAddrRx - The object's CAN Rx Address. short *iAddrTx - The object's CAN Tx Address. long *lSerialBaud - The Objects serial baud rate. short *iSerialPort - The Objects port. short *iDebugMode - Flag if in debug mode or not. In GetDeviceStatusEx only: short *sChannelNumber – CAN channel number. long lSper – Currently not in use.	**N/A**	Gets the status of a communication device. Check whether it is open, defined controller / Communication Protocol / Baud / Port / Addresses etc…

Method (Keyword)	Function Parameters	FlexDC Keyword	Function Description
GetFirstConnectedDeviceId	int *pHandle - Enumeration handle. long *pVal – Next connected device id in the system	**N/A**	Used to enumerate the device id connected in the system. This function returns the next device id connected in the system. Call this function only after calling GetFirstConnectedDeviceId.
GetNextConnectedDeviceId	int *pHandle - Enumeration handle. long *pVal – First connected device id in the system		
IsConnected/Ex	short sControllerType- Controller type the object is connected to. short sCOMMType- Communication type the object is connected to short sAddrRx- CAN Rx Address short sAddrTx - CAN Tx Address long lSerialBaud – Serial Baud Rate short sSerialPort – Serial Port In IsConnectedEx only: long lChannelNumber – CAN channel number long lSper – Currently not in use.	**N/A**	Returns the DevID of a connected configuration. If the configuration is not connected, -1 will be returned.
KillRepetative	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation.	**KR**	Kills the repetitive motion
LoadFlashParameters	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation.	**LV**	Loads parameters from SC flash

Method (Keyword)	Function Parameters	FlexDC Keyword	Function Description
OpenDevice/Ex	short sDevID– Opened Device ID short sControllerType – see section 23.2, Part V, for controller type values. short sCOMMType– see section 23.1, Part V, for communication type values. short sAddrRx – CAN Rx Address short sAddrTx– CAN Tx Address long lSerialBaud – Serial baud rate short SerialPort – Serial port (1-10) short DebugMode – Debug mode flag. In this case all parameters will return 0. In OpenDeviceEx only: long lChannelNumber – CAN channel number long lSper – Currently not used.	**N/A**	Attempts to open a communication device on defined controller / Communication Protocol / Baud / Port / Addresses etc…
ProgramDownload	short sDEVID– Opened Device ID BSTR bstrFileName- Macro file name (full path). int iTargetFWVersion – Firmware version number. int iTargeacroBufferOffset – Target macro buffer offset. int iTargeacroBufferSize – Target macro buffer size. int *iWarning – Number of warnings.	**N/A**	Download macro file (.scm) to controller.
ProgramClearNumberStack	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation.	**QZ**	Clears the program's number stack
ProgramDescriptionData	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation BSTR *bstrString- The returned program Description data.	**QU**	Returns a BSTR of the current description data in the SC program to bstrString. Note that bstrString is allocated by the server and must be released by client.
ProgramExecute	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation.	**QE**	Executes the program

Method (Keyword)	Function Parameters	FlexDC Keyword	Function Description
ProgramHalt	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation.	**QH**	Halts the running of the program
ProgramInitialize	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation.	**QI**	Initializes the program
ProgramKill	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation.	**QK**	Kills the running of the program
ProgramLoad	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation.	**QL**	Loads the program from the SC Flash
ProgramNumberStack	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation BSTR *bstrString- The returned program number Stack.	**QN**	Returns a BSTR of the current number stack in the SC to bstrString. Note that bstrString is allocated by the server and must be released by client.
ProgramStack	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation BSTR *bstrStack- The returned program stack String.	**QQ**	Returns a BSTR of the programs call stack to bstrStack. Note that bstrStack is allocated by the server and must be released by client.
ProgramTrace	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation.	**QT**	Traces a single line of the program

Method (Keyword)	Function Parameters	FlexDC Keyword	Function Description
ProgramUpload	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation BSTR *bstrString- The returned program Upload.	**QU**	Returns a BSTR of the current loaded program in the SC to bstrString. Note that bstrString is allocated by the server and must be released by client.
RecordingUpload	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation BSTR *bstrString- The returned Uploaded recording.	**UD**	Returns a BSTR of the current uploaded recording data from the SC to bstrString. Note that bstrString is allocated by the server and must be released by client.
RecordingUploadEx	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation long lTimeout – Max Time to wait till all data is in buffer. long lBufferLen – Recorded data buffer length. long lSper1 – First index of data to receive. long lSper2 – Last index of data to receive. VARIANT *variant – If communication is RS232 a BSTR string, else a long array.	**UD**	Returns a part or all the recording data buffer, the amount of data to be returned is specified in lSper2-lSper1 vars.
RecordingBegin	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation.	**BR**	Begin Recording.
ResetController	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation.	**RS**	Resets the controller

Method (Keyword)	Function Parameters	FlexDC Keyword	Function Description
SaveFlashParameters	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation.	**SV**	Saves savable parameters to SC flash
SendStringToController	short sDEVID– Opened Device ID BSTR stOutString- String to send to controller. BSTR *stInString- String returned from controller	**N/A**	Sends a string to the controller as is. Note that stInString is allocated by the server and must be released by client.
Stop	short sDEVID– Opened Device ID short sAxis– Defined Axis for operation.	**ST**	Stops Motion on defined axis
SendString	short sDEVID– Opened Device ID BSTR stOutString-String to send to opened Device. BSTR *stInString – String returned from opened Device.	**N/A**	Sends and gets a string to/from an opened device. The string is sent as-is and is returned as-is.
ShutDcomServer	N/A	**N/A**	Close the DCOM brutally.

Table 50: Interface Commands Methods

23 Object Definitions

This chapter defines the definitions that are needed for the various object parameters / methods. The definitions are divided into the following sections: Communication Protocols and Servo Controllers supported by the COM object codes.

23.1 Communication Protocols

Definition in 'h' file	Value	Parameter / Function used in
CAN_COMMUNICATION	20	OpenDevice OpenDeviceEx
RS232_COMMUNICATION	21	OpenDevice OpenDeviceEx

23.2 FlexDC Type Supported

Definition in 'h' file	Value	Parameter / Function used in
CRS_SC_2M_AT	54	OpenDevice OpenDeviceEx

Part VI – Communication Library (Commdll.dll)

24 Introduction

24.1 General

The "Comdll.dll" library enables the user and interface to COM ports, using the RS232 communication.

The functions included in this DLL interface have been customized to be used with Nanomotion products, and might not necessarily suite for other interface.

The FlexDC CD includes the following files in order to install the "Comdll.dll" library:

- Comdll.dll – the communication .dll file.
- Comdll.lib – the comdll.dll library file interface.
- Cdevapi.h – the software header file to include in the project.

This DLL is for Win32 based operating systems ONLY.

Part VI provides a description of all functions, exported by the "Comdll.dll" library module.

24.2 The RS232 Communication DLL

The Windows RS232 Communication Interface library is written in Microsoft Visual C++ version 6, and is designed to work in 32bit Windows Operating systems. In order to use the DLL interface, the LIB and header files should be included in the project. It is recommended to use the Microsoft Visual C++ development platform (Microsoft Developer Studio) versions 6 or further.

Notes:

- *This DLL supports only the RS232 protocol. CAN interfaces libraries depend on the actual CAN hardware used on the PC, thus no standard DLL is supplied. Nanomotion supports users to interface the FlexDC servo controller, using their own CAN hardware, upon request.*
- *Functions described in Part VI are for reference only. The functions may be subjected to changes by Nanomotion Ltd.*

25 The Communication Library - COMDLL.DLL

25.1 Instructions

The RS232 communication library is built as a Windows DLL (Dynamic Linked Library). The main library module file name is "Comdll.dll". In order to run applications that use this library, the user must implement the following steps:

1. Copy the "**Comdll.dll**" file to application directory, or to the Windows system directory.
2. The functions exported by the library, which provide an API (application program interface) to the FlexDC servo controller, are declared in the DLL include file, "**Cdevapi.h**". Therefore, the user must include this file in each source code file (refer to "Part III– FlexDC Macro Language") that tries to access DLL functions, using the following deceleration in the header of your implementation file:

   #include "cdevapi.h"
3. In order to link the project with the DLL at run time, the user must include the DLL library file "Comdev.lib" in the project. This library file informs the linker (during the link process) that the functions declared by the "Cdevapi.h" file are imported from the DLL at runtime (see the Terminal Demo Application project workspace).

The following section describes the DLL API functions.

25.2 DLL API Functions

25.2.1 CreateComDev

Function Name:

__declspec(dllexport) PVOID CreateComDev( );

Purpose:	This function creates a communication device module, and allocates memory inside the DLL for the communication process. The user must call this function from the application before calling any other function in the DLL.
Return Value:	A pointer to an internal Communication device object. Check that this pointer is not NULL. Use this pointer in all subsequent calls to DLL functions.

25.2.2 DestroyComDev

Function Name:

__declspec(dllexport) BOOL DestroyComDev(PVOID pComDev)

Purpose:	This function destroys the communication module created by the call to CreateComDev(), and frees all memory. The user must call this function before closing an application.
Input Arguments:	pComDev - pointer to Comm device, returned by CreateComDev().
Return Value:	TRUE if the function succeeds, FALSE otherwise.

25.2.3 SetupComDevInfo

Function Name:

__declspec(dllexport) int SetupComDevInfo

(PVOID pComDev, BYTE nPort, WORD wBaudRate ,BYTE bByteSize , BYTE bParity , BYTE bStopBits)

Purpose:	This function initializes the communication port device. Call this function before trying to open the port.
Input Arguments:	pComDev- Pointer to Communication device, returned by CreateComDev() function. nPort - Port Number: 1,2,3 or 4. wBaudRate - Baud Rate: CBR_300 , _4800, _9600_19200. bByteSize - Byte Size: Must be 8 bits. Bparity - Parity: Must Be NOPARITY bStopBits - Stop Bits: Must Be ONESTOPBIT
Return Value:	0 if OK, otherwise – an error occurred. Use the ShowCommErr() function with the return value to show an error message.

25.2.4 OpenComDev

Function Name:

__declspec (dllexport) BOOL OpenComDev (PVOID pComDev,
HWND hWnd,
DWORD dwInQueue,
DWORD dwOutQueue)

Purpose:	This function opens the selected communication device.	
Input Arguments:	pComDev	- Pointer to Communication device, returned by CreateComDev() function.
	hWnd	- HANDLE of window to receive notifications, if this parameter is NULL, no notification is sent.
	dwInQueue	- Received buffer maximum size.
	dwOutQueue	- Sent buffer maximum size.
Return Value:	TRUE if the function succeeds, otherwise – FALSE.	
	If the selected port is already open or used by another device, the function fails.	

25.2.5 CloseComDev

Function Name:

__declspec(dllexport) BOOL CloseComDev (PVOID pComDev)

Purpose:	This function closes an opens communication device.	
Input Arguments:	pComDev	- pointer to Comm device, returned by CreateComDev()
Return Value:	TRUE if the function succeeds, FALSE otherwise.	

25.2.6 IsConnected

Function Name:
__declspec(dllexport) BOOL IsConnected(PVOID pComDev)

Purpose:	Checks if port is already connected. Call this function to check if port is already connected.	
Input Arguments:	pComDev	- pointer to Comm device, returned by CreateComDev()
Return Value:	TRUE if the port is open. FALSE if port is closed.	

Note:

- *The user cannot use this function to check if a port is already used by another device. Using this function the user can check if a previous call to OpenComDev() is successful.*

25.2.7 CopyComInfo

Function Name:

__declspec(dllexport) BOOL CopyComInfo (PVOID pComDev,
COMDEVINFO *cinfo,
BOOL retrive)

Purpose:	Checks if port is already connected. Call this function to check if port is already connected. Copy COM info data structure from or to the dll COM info data structure, according to the retrieve flag. If the retrieve flag is TRUE, copies data from the dll to the given structure, otherwise copy data to the dll COM info structure.	
Input Arguments:	pComDev	- pointer to Comm device, returned by CreateComDev()
	cinfo	- pointer to COMDEVINFO structure.
	retrive	- flag indicate if to set or retrieve COMDEVINFO.
Return Value:	TRUE if the operation is OK.	

25.2.8 SendComString

Function Name:

__declspec(dllexport) int SendComString (PVOID pComDev,
LPSTR pInpBuf ,
DWORD nMaxLenInpBuf ,
DWORD &nBytsRead ,
LPSTR pOutBuf ,
DWORD nOutBufDataLen,
DWORD &nBytsSent ,
DWORD nTimeOut,
BOOL fWaitCursor ,
BOOL WaitResoponse
DWORD *pCommDelayTime)

Purpose: Send a string and optionally wait for a response from the client.

Input Arguments:

pComDev - pointer to Comm device, returned by CreateComDev()

pInpBuf - Pointer to Buffer Receiving RS232 Data

nMaxLenInpBuf- Max Size of Receiving Buffer

nBytsRead - Number of Bytes actually received from the communication port.

pOutBuf - Pointer to Write Data Buffer

nOutBufDataLen- Number of Data Bytes in pOutBuf

nBytsSent - Actual number of bytes sent (must be equal to nOutBufDataLen)

nTimeOut - Time out in miliseconds for wait.

fWaitCursor - Use Hourglass cursor while waiting.

WaitResoponse- Flag whether to wait for response from controller.

pCommDelayTime- Communication response delay time.

Return Value: 0 if function succeeds. If the function fails, an error code is returned.

25.2.9 ReadString

Function Name:
__declspec(dllexport) int ReadString (PVOID pComDev,
LPSTR plnpBuf ,
DWORD nMaxLenInpBuf ,
DWORD &nBytsRead)

Purpose:	Read string from the COM port.
Input Arguments:	pComDev - pointer to Comm device, returned by CreateComDev()
	pInpBuf - Pointer to Buffer Receiving RS232 Data
	nMaxLenInpBuf- Max Size of Receiving Buffer
	nBytsRead - Number of Bytes actually received from the communication port.
Return Value:	0 if function succeeds. If the function fails, an error

25.2.10 ReadComBuf

Function Name:

__declspec(dllexport) int ReadComBuf (PVOID pComDev,
BOOL bBlock,
LPSTR pInpBuf ,
DWORD nMaxLenInpBuf ,
DWORD &nBytsRead)

Purpose:	Reads a string from the COM port with option for blocking the incoming read thread.	
Input Arguments:	pComDev	- Pointer to Comm device, returned by CreateComDev()
	bBlock	- Block the read thread.
	pInpBuf	- Pointer to Buffer Receiving RS232 DatanMaxLenInpBuf- Max Size of Receiving Buffer
	nBytsRead	- Number of Bytes actually received from the communication port.
Return Value:	0 if function succeeds. If the function fails, an error code is returned.	

25.2.11 ReadSizeComBuf

Function Name:

__declspec(dllexport) int ReadSizeComBuf (PVOID pComDev,
LPSTR plnpBuf ,
DWORD nMaxLenInpBuf ,
long lReadNumberOfBytes,
DWORD dwTimeOut,
DWORD &nBytsRead)

Purpose: Reads a number of bytes from the port within a given timeout.

Input Arguments:

pComDev - Pointer to Comm device, returned by CreateComDev()

pInpBuf - Pointer to Buffer Receiving RS232 Data

nMaxLenInpBuf - Max Size of Receiving Buffer

lReadNumberOfBytes - Expected number of bytes to read from port.

dwTimeOut - Time out In miliseconds for wait.

nBytsRead - Number of Bytes actually received from the communication port.

Return Value: 0 if function succeeds. If the function fails, an error code is returned.

26 Communication Error Codes

The following error codes are reported by the DLL:

COM_OK = 0	No Error
COM_TIMEOUT	Time-out while waiting for response
COM_COM_ERR	Communication Error
COM_READ_ERR	Communication Device Read Error
COM_WRITE_ERR	Communication Device Write Error
COM_BUFFER_TOO_BIG	Input/output Buffs to big (255 Max)
COM_BUFFS_NOT_VALID	Input or Output buffers are NULL.
COM_PDEVNOT_VALID	Pointer to communication device is NULL.
COM_DEVNOT_VALID	Never returned value.
COM_PARAM_ERR	Error in Communication parameters.
COM_BAD_ECHO	Bad Echo received from controller.
COM_CANT_PURGE_COM	Communication purging device error.
COM_NOTIMPLEMENTED	Function Not Implemented.

27 Code Example

The following example opens the device, sends and receives a string to and from a client, and closes the device:

```
//
// create the com device and check if error.
//
void *pComDev = CreateComDev() ;
if(pComDev == NULL)
{
        return Error ;
}

//
// set device parameters.
//
COMDEVINFO comInfo ;
comInfo.idComDev            = NULL          ;
comInfo.bPort          = 1              ;
comInfo.fConnected          = FALSE         ;
comInfo.fXonXoff       = FALSE          ;
comInfo.bByteSize      = 8              ;
comInfo.bFlowCtrl      = 0              ;
comInfo.bParity     = NOPARITY              ;
comInfo.bStopBit       = ONESTOPBIT  ;
comInfo.wBaudRate           = CBR_38400             ;
comInfo.fReConnect          = FALSE         ;
comInfo.cEvtChar       = '>'            ;
comInfo.iSpecialVersion        = 0                      ;

//
// Set Com Info Data and Check if OK
//
if(!CopyComInfo(&comInfo,FALSE) )
{
        return Error;
}
```

```
//
// open the device.
//
DWORD dwInQueue                 = 250 ;
DWORD dwOutQueue    = 250 ;
If(!OpenComDev(pComDev,NULL, dwInQueue, dwOutQueue))
{
            return Error ;
}

//
// check if connected.
//
If(!IsConnected(pComDev))
{
          return Error ;
}

//
// send and get data from device.
//
char pOutBuf[16]      = “XSP=1000\r”          ;
char pInpBuf[255]                             ;
DWORD dwNBytesRead            = 0                       ;
DWORD dwnOutBufDataLen      = 9           ;
DWORD dwnBytesSent            = 0           ;
int           iRc               = 0       ;

iRc = SendComString(    pComDev  ,
                          pInpBuf, 255, dwNBytesRead ,
                          pOutBuf, dwnOutBufDataLen, dwnBytesSent,
                          500               , // wait for .5 sec
                          FALSE       ,
                          TRUE            ) ; // do wait for answer.
//
// Check Error Code
//
If(iRC != COM_OK)
{
          return Error ;
}

//
// now the answer is in pInpBuf. destroy com device (clean up).
//
DestroyComDev(pComDev) ;
pComDev = NULL ;
```

Part VII – Glossary

28 Glossary

The following definitions are used provided only for this user manual.

- **Abort Input:** a dedicated digital input typically connected to the machine's emergency button. When the FlexDC detects an active state at this input it immediately disables both motors. In addition to the standard firmware support for the Abort signal as noted above, the FlexDC also supports the Abort signal. It is monitored by the Hardware to disable all drivers in case Abort is sensed.
- **Clause:** a single, complete, independent communication statement that can be interpreted and evaluated. Each clause consists of keywords and operators and is terminated by a terminator (to identify end of clause).
- **Clause – Assignment:** a communication statement sent by a host and instructs the FlexDC to assign a value to a specified parameter. A typical assignment clause consists of: Keyword "=" value terminator.
- **Clause – Command:** a communication statement sent by a host and instructs the FlexDC to perform a specified command (process). A command clause consists of: Keyword terminator.
- **Clause – Report:** a communication statement sent by a host and instructs the FlexDC to report the value of a specified parameter. A typical report clause consists of: Keyword terminator.
- **Clause – Terminator:** character that identifies end of communication clause. It can be <CR> or ";" in the communication from a host to an FlexDC or ">" in the opposite direction (all for the RS232 line).
- **Command Interpreter:** the Commands Interpreter is an internal software module of the FlexDC firmware, responsible for interpreting Clauses sent to the controller. The Command Interpreter handles all commands passed to the FlexDC.
- **Communication Protocol:** the low-level hardware and software definition of a communication channel. In RS232, for example, it includes the baud-rate, handshake options, parity, etc.
- **Communication Syntax, Language Syntax:** the rules that define the correct sequence of characters that may create a valid communication clause.

- **Digital Control Filter:** an algorithm that is periodically executed (8192 times per second). The algorithm compares the desired motor position and its actual position to calculate a command to the motor to minimize the difference between these values. The FlexDC Digital Control Filter algorithm supports both standard position based PIV, as well as Position Over Velocity loop structure. The FlexDC supports additional advanced features. Refer to chapter 4, Part II, for further information.
- **Echo:** in RS232 mode, the FlexDC automatically echoes (send a copy back) each character that it receives during normal communication. The returned character can be used by the host to verify proper communication. In the binary CAN bus communication protocol, ECHO is not supported. Only OK/ERR prompt is used.
- **Error Codes:** in case that the FlexDC encounters an error when interpreting a received clause it ignores this clause and responds with "?" before the returned terminator (">"). The FlexDC also stores a code for the interpretation error at a parameter named "EC" – which can be later reported to analyze the error source. A separate parameter "QC" holds the error codes of any program running in the controller (Scripts or Macro).
- **Fault Input:** a dedicated digital input whose source is typically the motor's driver. It is used to inform the FlexDC about a driver's malfunction – for which the FlexDC needs to inhibit the driver and to abort all motion activities.
- **Firmware Version Downloading:** the FlexDC executes an internal firmware (BIOS) to perform all its tasks. From time to time new firmware versions are released (corrections of problems, new features, etc.). New firmware versions are supplied by Nanomotion (or available on our web site). The FlexDC, together with the Nanomotion Shell Apllication, enables the downloading of a new version via the RS232 (ONLY) line. The advantages of this process (over older EPROM replacement method) are clear.

- **Flash Memory (Flash):** the FlexDC includes a 2M[bits] Flash Memory for its firmware, parameters and user program. The Flash Memory is, in principal, similar to an EEPROM memory. It enables the downloading of a new firmware version.
- **Host:** a computer, terminal, PLC or any other device which may send communication clauses to the FlexDC, via one of its communication links.
- **Identifiers – Axes:** the FlexDC Commands Syntax always requires an axis identifier before the keyword itself. If a Keyword attribute is non-axis related, any axis identifier is legal, and has the same result. The Command Interpreter ignores the axis identifiers of non-axis-related keywords.
- **Identifiers – Group Axes:** the FlexDC Commands Syntax supports the concept of Axes Group identifier definition. An Axes Group allows the user to define an arbitrary sub-set of controller axes to be acted upon[15]. Like in normal axes identifiers, the Command Interpreter ignores the Group Identifier of non-axis-related keywords. The FlexDC supports the 'B' – Both group identifier only.
- **Inhibit Output:** a dedicated digital output of the FlexDC (one for each axis) which is used to enable/disable an external motor's driver. The inhibit output reflects the state of the MO parameter.
- **Incremental Encoder:** a standard position sensor used as a position feedback in conjunction with motors and servo systems. A special FlexDC hardware circuit uses the encoder's signals to continuously sense the motor/load position (and speed) and to accordingly control the motor motion.
- **Keyword:** a token, consisting of two characters, which identifies a unique FlexDC command or parameter.
- **Keyword Attributes:** each Keyword of the FlexDC has one or more attributes. The Keyword attributes tell the command Interpreter how to be treated. For example, a Keyword can be an axis related Keyword (related to an axis) or Global Keyword.

[15] The current FlexDC firmware does not support configurable Axis Groups.

- **Limit – Hardware RLS, FLS:** most (although not all) motion systems have mechanical end-of-travel stops (especially with linear load motion). In order to prevent the load from hitting these stops, an electronic device/switch is located before each stop (Reversed and Forward) to detect this situation. These switches are connected to the FlexDC RLS and FLS digital inputs (Reverse Limit Switch and Forward Limit Switch). When the FlexDC detects an active state at one of these inputs it stops any motion toward the related direction.
- **Limits – Software HL, LL:** similarly to the hardware limits (RLS and FLS above), the FlexDC supports software limitation for motion range. HL (High Limit) and LL (Low Limit) defines a position range in which the FlexDC operates normally. Whenever the motor's position exceeds this range, the FlexDC stops any motion to the related direction.
- **Motion – Modes:** motion Mode defines the method in which the FlexDC calculates the desired position command as a function of time. The FlexDC supports various motion modes. The basic modes are listed below:
  - Point To Point (PTP).
  - Jogging.
  - Gearing.
  - Step.
  - Repetitive Step and PTP.
- **Motion – Profiling:** motion Profiling is the actual algorithm that calculates new reference points to the servo loop according to the selected Motion Mode.
- **Motion - On-The-Fly Changing:** a characteristic of the FlexDC that enables the modification of most of its parameters even when they are active. For example, the PIV parameters can be modified while the motor is in servo loop (motor is ON). A unique characteristic of the FlexDC is that all (except profile smoothing) of its motion parameters (such as: speed, acceleration, deceleration, distance, etc.) can be modified on-the-fly under almost any conditions.

- **Position Capture Events:** Capture Position feature is the ability of the encoder interface hardware to capture (latch) the exact encoder location when a pre-defined Input or encoder Index is detected. The Capture hardware can latch encoder position when counting at ANY encoder speed. The Capture mechanism can be programmed to latch encoder positions based on a user defined digital input, or encoder index pulse.
- **Position Compare Events:** the Compare Position feature is the ability of the encoder interface hardware to compare the actual encoder hardware counter value to a pre-defined user register value, and to generate a H/W pulse when there is a condition match. The basic compare mechanism can work at ANY encoder speed. Compare mechanism can be operated as a fixed GAP auto increment condition, or variable GAP tables.
- **Scripts or Macro Programming:** the FlexDC supports up to 2 simultaneous internal programs (also referred to as “Scripts” or “Macro” programs). Internal programs are used for tasks like Homing an axis, or other user defined low level servo tasks. The FlexDC is provided with an advanced SDE (“Software Development Environment”), including very powerful debugger and editor utilities, making Scripts programming and debugging an easy task.
- **Windows Shell Program, Nanomotion Shell Application:** Nanomotion provides an enhanced Windows 9x (or NT/2000/XP) Nanomotion Shell Application program for easy and fast interface with the FlexDC. Using the Nanomotion Shell Application, starting-up or verifying a new idea/concept is just few mouse clicks away.