Grasper Control Language (GCL) Protocol

The Grasper Control Language (GCL) is available and licensed to the public, free of charge, by Barrett Technology, Inc. The GCL development was staffed by Yoky Matsuoka, PhD, Daniel Griscom, William Townsend, PhD, and others at Barrett Technology, Inc.

Until recently there was no comprehensive control language for dexterous graspers. So with the rise of dexterous graspers, it became important to develop an expandable GCL whose initial vocabulary and grammar are clear and simple to implement. In addition, two (2) control modes are essential: supervisory (where commands are interpreted at the grasper) and realtime (where the control loops are defined and implemented on a separate computer or robot controller outside the grasper). The language also allows for fast, microsecond switching between the two modes.

Grasper Control

This article explains the command structure of the Grasper Control Language (GCL) to communicate with a grasper.

Supervisory Control

The GCL can be used in either of two modes:

1. high-level Supervisory mode
2. low-level RealTime mode.

Supervisory mode allows you to command individual or multiple motors to close, open and move to specific positions. You also have access to all of the parameters. This set of commands is commonly used for most grasping situations. If real-time control of the motor position, velocity, or strain is needed, use the RealTime control.

Supervisory mode accepts commands from the user program and will not return control of the grasper until the command is finished being processed. The grasper expects valid commands and will return a status code for an invalid command or if another problem occurs. When the command is finished being executed, all status codes and requested information have been sent, the hand will return the command prompt "=>". At this point, you can send another command.

Command Structure

Firmware resides on the grasper and interprets the commands it receives. This command structure allows you to build the desired command easily. The format is as follows:

<Motor><Command> <Parameter> <Value>

Following is a list of the <*Motor*> prefixes:

Table 1 - Motor Prefixes

Value	**Motor**
1	Finger F1
2	Finger F2
3	Finger F3
4	Spread
G	Finger F1, Finger F2, Finger F3
S	Spread
<No Motor Specified>	Finger F1, Finger F2, Finger F3, Spread

Note: Any combination of motor prefixes can be used together to produce the desired result. Example: 12<Command> <Parameter> <Value> will activate Fingers F1 and F2.

Firmware Parameters

Parameter: **ACCEL**
Purpose: Acceleration value for position control.
Values: 0 - 32767
Default: *Grasp:* *1*

Spread: *1*

Parameter: **BAUD**
Purpose: Returns the current baud rate of the hand divided by 100.
Values: 6, 12, 24, 48, 96, 192 and 384
Default: *96*

Notes: *The value returned is in hundreds of bytes per second. To determine the actual baud rate, multiply the value returned by 100.*

Parameter: **DP**
Purpose: This parameter defines the default position for a move command.
Values: 0 - 20000 encoder counts
Default: *150 (Spread), 1000 (Fingers)*

Notes: *None.*

Parameter: **DS**
Purpose: This parameter defines default step sizes for incremental open and close commands.
Values: 0 - 20000 encoder counts
Default: *150 (Spread), 1200 (Fingers)*

Notes: *None.*

Parameter: **EN**
Purpose: Specifies if a motor should be selected when a command has no prefix.
Values: TRUE (selected), FALSE (not selected)
Default: *Grasp:* *TRUE*

Spread: *TRUE*

Notes: *When a close command is issued, C, with no motor prefixes, all motors will close with the default values.*

Parameter: **FDZ**
Purpose: Derivative zero value for the motor control filter.
Values: 0 - 255
Default: *Grasp:* *221*

Spread: *221*

Parameter: **FIP**
Purpose: Integral pole value for the motor control filter.
Values: 0 - 255
Default: *Grasp: 66*

Spread: 66

Parameter: **FPG**
Purpose: Proportional gain value for the motor control filter.
Values: 0 - 255
Default: *Grasp: 200*

Spread: 100

Parameter: **HOLD**
Purpose: Specifies if a motor should hold position when idled.
Values: TRUE (hold position), FALSE (do not hold position)
Default: *Grasp: FALSE*

Spread: TRUE

Notes: *Because the fingers are not backdrivable when the motors are idled they will not be able to move freely. However, because the spread is backdrivable it requires this parameter be TRUE to hold its position when idled.*

Parameter: **LCPG**
Purpose: This flag specifies if the RealTime control block contains control proportional gain.
Values: FALSE (does not contain), TRUE (does contain)
Default: *FALSE*

Notes: *Motor command = (LCPG / 4) * (Control Velocity - Actual Velocity)*

Parameter: **LCV**
Purpose: This flag specifies if the RealTime control block contains control velocity.
Values: FALSE (does not contain), TRUE (does contain)
Default: *TRUE*

Notes: *The size of the control velocity should be 1 signed byte.*

Parameter: **LCVC**
Purpose: LCV is multiplied by the control velocity coefficient (LCVC) to determine the control velocity.
Values: 0 - 255
Default: *1*

Notes: *Control velocity = LCV * LCVC*

Parameter: **LFAP**
Purpose: This specifies if the RealTime feedback block contains the feedback absolute position.
Values: FALSE (does not contain), TRUE (does contain)
Default: *TRUE*

Notes: *The size of the feedback absolute position should be an unsigned 2-byte word.*

Parameter: **LFDP**
Purpose: This flag specifies if the RealTime feedback block contains the feedback delta position.
Values: FALSE (does not contain), TRUE (does contain)
Default: *FALSE*

Notes: *The size of the feedback delta position should be 1 signed byte.*

Parameter: **LFDPC**
Purpose: The actual change in position is divided by feedback delta position coefficient (LFDPC) to determine LFDP.
Values: 0 - 255
Default: *1*

Notes: Delta position is the change in position from the last reported position and is limited to one signed byte. The current position is read and compared to the last reported position. The difference is divided by the RealTime variable LFDPC, clipped to a single signed byte, and then sent to the host. The value sent to the host should be multiplied by LFDPC and then added to the last reported position.

Parameter: **LFS**
Purpose: This specifies if the RealTime feedback block contains the feedback strain gage value.
Values: FALSE (does not contain), TRUE (does contain)
Default: *TRUE*

Notes: *The size of the feedback strain gage value should be 1 unsigned byte.*

Parameter: **LFV**
Purpose: This specifies if the RealTime feedback block contains feedback velocity.
Values: FALSE (does not contain), TRUE (does contain)
Default: *TRUE*

Notes: *The size of the feedback velocity should be 1 signed byte. The actual velocity is LFC*LFVC.*

Parameter: **LFVC**
Purpose: Actual velocity is divided by feedback velocity coefficient (LFVC) to determine LFV.
Values: 0 - 255
Default: *1*

Notes: *On the host computer the actual velocity of the motors is equal to LFV * LFVC.*

Parameter: **MCV**
Purpose: This parameter defines the maximum close velocity.
Values: 0 - 255
Default: *35 (Spread), 65 (Fingers)*

Parameter: **MOV**
Purpose: This parameter defines the maximum open velocity.
Values: 0 - 255
Default: *35 (Spread), 55 (Fingers)*

Notes: *The minimum velocity required to reset the TorqueSwitch™ and open and close the fingers is 40.*

Parameter: **MPE**
Purpose: Maximum position error allowed for a commanded position.
Values: 0 - 30,000
Default: *Grasp:* *25*

Spread: *25*

Notes: *If the final position is not within +/- MPE encoder counts of the desired position then the hand will return an error.*

Parameter: **MSG**
Purpose: This parameter defines the maximum strain gage value before the motor is stopped.
Values: 0 - 256
Default: *256*

Notes: *Setting the value to 256 indicates that the strain gage value will never stop the motors.*

Parameter: **P**
Purpose: This parameter specifies the present motor position.
Values: 0 - 20000 encoder counts
Default: *N/A*

Notes: *This parameter cannot be set.*

This section lists all of the firmware parameters and their values for a grasper.
Parameter: **S**
Purpose: This parameter defines the current state of the motor.
Values: 0 (motor found and initialized) or 1 (motor not initialized)
Default: *N/A*

Notes: *This parameter can not be set.*

Parameter: **SG**
Purpose: This parameter specifies the current strain gage value.
Values: 0 - 255
Default: *N/A*

Notes: *This parameter can not be set.*

Parameter: **SGFLIP**

Purpose: Specifies if the reported strain should be (255 - actual strain).

Values: TRUE (reported strain = (255 - actual strain)), FALSE (reported strain = actual strain)

Default: *Grasp:* *FALSE*

Spread: *N/A*

Notes: *Setting this value will inverse the direction of the change in strain for a given torque.*

Parameter: **TEMP**

Purpose: Returns the present temperature on the CPU board in tenths of degrees Celsius.

Values: -550 to 1250

Default: *N/A*

Notes: *The value returned is in tenths of degrees. To determine the actual temperature, divide the value by 10.*

Parameter: **TSTOP**

Purpose: Time in milliseconds before the motor is considered stopped.

Values: 0 - 32767

Default: *Grasp:* *30*

Spread: *30*

Notes: *None.*

Firmware Commands

Command: **C**
Function: Closes specified motors.
Parameters: N/A
Notes: *None.*

Command: **ERR**
Function: Returns a description of the status code specified.
Parameters: Status code numbers.
Notes: *Does not take motor prefixes.*

Command: **FDEF**
Function: Loads the factory default values of the parameters from EEPROM into memory.
Parameters: N/A
Notes: *This command loads the following parameters: MCV, MOV, DS, MSG, DP, FPG, FIP, FDZ, EN, SGFLIP, ACCEL, MPG, TSTOP, HOLD, LCV, LCVC, LCPG, LFV, LFVC, LFS, LFAP, LFAP, LFDPC.*

Command: **FGET**
Function: Gets the specified parameters.
Parameters: MOV, MCV, MSG, DS, DP, LCV, LCVC, LCPG, LFV, LFVC, LFS, LFAP, LFDP, LFDPC, FPG, FIP, FDZ, ACCEL, MPE, TSTOP, HOLD, SGFLIP, EN, BAUD, S, P, SG

Command: **FLISTA**
Function: Lists all of parameters and their read/write status.
Parameters: N/A
Notes: *Does not take motor prefixes.*

Command: **FLISTAV**
Function: Lists all of the present parameter values.
Parameters: N/A
Notes: *The parameters are listed in the same order they are displayed by the command FLISTA. This command does not take motor prefixes.*

Command: **FLOAD**
Function: Loads the saved parameters from EEPROM into memory.
Parameters: N/A
Notes: *This command loads the following parameters: MCV, MOV, DS, MSG, DP, FPG, FIP, FDZ, EN, SGFLIP, ACCEL, MPG, TSTOP, HOLD, LCV, LCVC, LCPG, LFV, LFVC, LFS, LFAP, LFAP, LFDPC.*

Command: **FSAVE**
Function: Saves the present values of the parameters to EEPROM.
Parameters: N/A
Notes: *This command saves the following parameters: MCV, MOV, DS, MSG, DP, FPG, FIP, FDZ, EN, SGFLIP, ACCEL, MPG, TSTOP, HOLD, LCV, LCVC, LCPG, LFV, LFVC, LFS, LFAP, LFAP, LFDPC.*

Command: **FSET**
Function: Sets the specified parameters to the desired value.
Parameters: MOV, MCV, MSG, DS, DP, LCV, LCVC, LCPG, LFV, LFVC, LFS, LFAP, LFDP, LFDPC, FPG, FIP, FDZ, ACCEL, MPE, TSTOP, HOLD, SGFLIP, EN, BAUD

Command: **HI**
Function: Initializes the finger and spread motors. Opens all of the joints to their full open position and sets it to be zero.
Parameters: N/A
Notes: *The grasper will vibrate the joints during this initialization operation. This command needs to be executed before any motion commands.*

Command: **IC**
Function: Incremental close for specified motors.
Parameters: N/A
Notes: *The increment size is defined in the parameter DS.*

Command: **IO**
Function: Incremental open of specified motors.
Parameters: N/A
Notes: *The increment size is defined in the parameter DS.*

Command: **LOOP**
Function: Enters RealTime mode for the specified motors
Parameters: N/A

Command: **M**
Function: Move the specified motors to specified position
Parameters: Motor position, 0 - 20000
Notes: *If no position is given it will move to the value stored in DP.*

Command: **O**
Function: Opens specified motors.
Parameters: N/A
Notes: *None.*

Command: **PGET**
Function: Gets the parameter specified.
Parameters: TMP
Notes: *Does not take motor prefixes.*

Command: **RESET**
Function: Resets the grasper and loads all of the saved parameters from EEPROM.
Parameters: N/A
Notes: *Does not take motor prefixes. The motors need to be reinitialized before commanding motion after using this command. See the command FSAVE for parameter information.*

Command: **T**
Function: Stops actuating the motors.
Parameters: N/A
Notes: *None.*

Command: **VERS**
Function: Greeting message, shows the version number and the company contact information.
Parameters: N/A
Notes: *None.*

Command: **?<Command>**
Function: Help information about the <Command> specified.
Parameters: N/A
Notes: *None.*

Command: **^C**
Function: Stops the motors and clears the input buffer. A new prompt will be output.
Parameters: N/A
Notes: *None.*

RealTime Control

One features of the Grasper Control Language is RealTime control. This control mode allows you to send commands and receive feedback continuously from the grasper. Any desired control law can be applied by using the host computer to determine the desired motor command and then applying that command to the Grasper in real-time. The communication bandwidth is dependent on the amount of control information sent, feedback information requested and the selected baud rate.

Data from the host computer to the hand is grouped into control and feedback blocks. Each block has a single byte header, followed by a set of data. The control block header specifies whether or not control data is to follow, and whether or not a feedback block is to be returned. The feedback block header returned acknowledges the receipt of the control block or indicates an error. The control block header can also terminate the loop mode.

The possible control block header bytes are:

"C":	Control data follows; respond with a feedback block
"c":	Control data follows; respond with an acknowledgement character ("*")
"A":	No control data follows; respond with a feedback block
"a":	No control data follows; respond with an acknowledgement character
"^C":	Terminate loop mode

The possible feedback block header bytes are:

"*":	The Grasper has received the control block successfully.
"<CRLF>ERR":	An error occurred, the status code will follow immediately.

Before sending information to the Grasper in RealTime mode, it is necessary to determine what the control and feedback blocks will contain. Do this by setting the RealTime control flags before entering RealTime mode. Setting a flag TRUE indicates that it will be part of the control or feedback block. A flag for each motor needs to be set. Set the flags by using the FSET command. See Table 2 for a detailed description of the flags.

There are also three RealTime variables that need to be set before entering RealTime mode. These three variables affect how the RealTime control values are interpreted. Set these variables by using the FSET command. See Table 2 for a detailed description of the variables.

Table 2 - RealTime Control Parameters

Parameter	Name	Type	Function	Size in Block
LCV	Loop Control Velocity	Flag	If True, RealTime control block will contain control velocity	1 signed byte
LCVC	Loop Control Velocity Coefficient	Variable (integer)	LCV is multiplied by LCVC to determine control velocity	N/A
LCPG	Loop Control Proportional Gain	Flag	If True, RealTime control block will contain Proportional Gain	1 unsigned byte
LFV	Loop Feedback Velocity	Flag	If True, RealTime feedback block will contain feedback velocity	1 signed byte
LFVC	Loop Feedback Velocity Coefficient	Variable (integer)	Actual velocity is divided by LFVC to get LFV	N/A
LFS	Loop Feedback Strain	Flag	If True, RealTime feedback block will contain strain information	1 unsigned byte
LFAP	Loop Feedback Absolute Position	Flag	If True, RealTime feedback block will contain absolute position	2 unsigned bytes
LFDP	Loop Feedback Delta Position	Flag	If True, RealTime feedback block will contain delta position	1 signed byte
LFDPC	Loop Feedback Delta Position Coefficient	Variable (integer)	The actual delta position is divided by this to get LFDP	N/A

Now that all of the flags and variables have been set, it is time to begin RealTime control. Send the command <Motors>LOOP to enter RealTime mode. At this point the Grasper will respond with a "*" to acknowledge the start of RealTime control. It is now up to the host computer to build control blocks and send them to the Grasper.

Example:
This application uses fingers F1 and F2, and the spread. The fingers will receive velocity control information and report strain and delta position. The spread will just report delta position. All relevant coefficients will be set to 1.

Set the RealTime flags and variables by using the following commands:
12FSET LCV 1 LCVC 1 LCPG 0 LFV 0 LFS 1 LFAP 0 LFDP 1 LFDPC 1

4FSET LCV 0 LCVC 1 LCPG 0 LFV 0 LFS 0 LFAP 0 LFDP 1 LFDPC 1

Enter RealTime control by issuing the following command.
124LOOP
The Grasper will then send a single "*" and wait for control blocks. Each control block will consist of three bytes:

"C" [Control data follows; respond with feedback block]
1 signed byte of velocity for motor F1
1 signed byte of velocity for motor F2

Each feedback block will consist of six bytes:
"*" acknowledge character
1 unsigned byte of strain for motor F1
1 signed byte of delta position for motor F1
1 unsigned byte of strain for motor F2
1 signed byte of delta position for motor F2
1 signed byte of delta position for motor 4

Each control block from the host will stimulate a feedback block from the Grasper. When the host is finished, it will send the single character ^C (0x03); the Grasper will respond by printing the command prompt "=>", and waiting for a new command.

Status Codes

Status codes, see Table 3, are sent by the Grasper when the communication was successful, but the Grasper encountered a problem. Keep in mind that Grasper status codes are powers of 2, so the return value may encode multiple flags. Example: a status code of 3, indicates status code 2 and status code 1.

Table 3 - Hand Status Codes

Hand Status Code	*Description*
1	No motor board found
2	No motor found
4	Motor not initialized
8	not used
16	Couldn't reach position
32	Unknown command
64	Unknown parameter name
128	Invalid value
256	Tried to write a read only parameter
512	Timeout
1024	Too many arguments for this command
2048	Invalid RealTime control block header
4096	Command can't have motor prefix

Example Programs

Supervisory Mode Example Program

The following program is an example that shows how to program the Grasper in Supervisory mode using the C-Function Library. The code was generated using the BHControl Interface and compiled using Microsoft Visual C++ v6.0. This program initializes the Grasper and then opens and closes the grasp.

```
///////////////////////////////////////////////////////////
//                                                       //
//           Automatically Generated C++ Code            //
//           BHand Control Center Version 1.0            //
//                                                       //
//                   Supervisory Mode                    //
//                                                       //
///////////////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include "BHand.h"

BHand  bh;           // Handles all hand communication
int    value;        // Hand parameter obtained with Get command
int    result;       // Return value (error) of all BHand calls
```

```
////////////////////////////////////////////////////////////
// Error Handler - called whenever result!=0

void Error(void)
{
      printf( "ERROR: %d\n%s\n",result,bh.ErrorMessage(result));
      exit(0);
}


////////////////////////////////////////////////////////////
//  Initialize hand, set timeouts and baud rate

void Initialize(void)
{
      if(result=bh.InitSoftware(1,THREAD_PRIORITY_TIME_CRITICAL))
            Error();
      if( result=bh.ComSetTimeouts(0,100,15000,100,5000) )
            Error();
      if( result=bh.Baud(9600) )
            Error();
      if( result=bh.InitHand("") )
            Error();
}


////////////////////////////////////////////////////////////
//  Execute commands, return 1 if interrupted with a key

int Execute(void)
{
      printf( "Press Any Key to Abort..." );

      // Initializes all motors
      if( result=bh.InitHand( "123S" ) )
            Error();
      if( _kbhit() )
            { _getch(); return 1; }

      // Closes fingers F1, F2 and F3
      if( result=bh.Close( "123" ) )
            Error();
      if( _kbhit() )
            { _getch(); return 1; }

      // Opens fingers F1, F2 and F3
      if( result=bh.Open( "123" ) )
            Error();
      if( _kbhit() )
            { _getch(); return 1; }
      return 0;
}


////////////////////////////////////////////////////////////
//  Main function - initialize, execute

void main(void)
{
      printf( "Initialization..." );
      Initialize();
      printf( " Done\n" );

      printf( "Executing - " );
      Execute();
      printf( " Done without interruption\n" );
}
```

RealTime Mode Example Program

The following program is an example that shows how to program the hand in RealTime mode using the C-Function Library. The code was generated using the BHControl Interface and compiled using Microsoft Visual C++ v6.0. This program will close finger one and starts closing finger two when finger one reaches position 5000. Finger three starts closing when finger two reaches position 5000. The program is terminated after six seconds.

```
///////////////////////////////////////////////////////////
//                                                       //
//           Automatically Generated C++ Code            //
//           BHand Control Center Version 1.0            //
//                                                       //
//                   RealTime Mode                      //
//                                                       //
///////////////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include "BHand.h"

BHand  bh;           // Handles all hand communication
int    value;        // Hand parameter obtained with Get
int    result;       // Return value (error) of all BHand calls


///////////////////////////////////////////////////////////
// Error Handler - called whenever result!=0

void Error(void)
{
      printf("ERROR: %d\n%s\n", result, bh.ErrorMessage(result));
      exit(0);
}

///////////////////////////////////////////////////////////
//  Initialize hand, set timeouts and baud rate

void Initialize(void)
{
      if(result=bh.InitSoftware(1,THREAD_PRIORITY_TIME_CRITICAL))
            Error();
      if( result=bh.ComSetTimeouts(0,100,15000,100,5000) )
            Error();
      if( result=bh.Baud(9600) )
            Error();
      if( result=bh.InitHand("") )
            Error();
}

////////////////////////////////////////////////////////////
//  Set parameters, allocate data buffers, load files

void PrepareRealTime(void)
{
      // Set RealTime Flags to be sent during RealTime control
      if(result=bh.RTSetFlags("123", 1, 1, 0,  0, 1, 0, 1, 0, 1))
            Error();
}


////////////////////////////////////////////////////////////
//  Run RealRime loop, return 1 if interrupted with a key

int RunRealTime(void)
{
      double var[4][3];
      int N=0, motor;
      DWORD  time, tmstart;
      bool terminate=false;

      // Start RealTime Mode
      bh.RTStart( "123" );

      // Start timer
      tmstart = GetTickCount();

      // Send RealTime control to hand
      bh.RTUpdate();

      printf( "Press Any Key to Abort..." );

      // Control Hand until termination
      while( !terminate && !_kbhit() )
      {
            time = GetTickCount() - tmstart;

            // Get RealTime Position and time
            for( motor=0; motor<4; motor++ )
            {
                  // Get motor position
                  var[motor][0] = bh.RTGetPosition( motor+'1' );
                  // Get time
                  var[motor][1] = (double)time;
                  // Get number of iterations
                  var[motor][2] = (double)N;
            }
```

```
                // Set F1 close velocity to 55
                value = (int)(55.00);
                bh.RTSetVelocity( '1', value );

                // If F1 position is > 5000 then set F2 close
                // velocity to 55, otherwise set to 0
                value=(int)(((var[0][0])>(5000.00))?(55.00):(0.00));
                bh.RTSetVelocity( '2', value );

                // If F2 position is > 5000 then set F3 close
                // velocity to 55, otherwise set to 0
                value=(int)(((var[1][0])>(5000.00))?(55.00):(0.00));
                bh.RTSetVelocity( '3', value );

                // If the time is greater than 6 seconds, then stop
        // controlling hand in RealTime
                terminate = (0<(int)((var[0][1]) > (6000.00)));

                // Increment iterations
                N++;

                // Send all updated control parameters to the hand
                bh.RTUpdate();
        }

        // Exit RealTime mode
        bh.RTAbort();
        if( _kbhit() )
                { _getch(); return 1; }
        else
                return 0;
}

////////////////////////////////////////////////////////////
//  Main function - initialize, execute

void main(void)
{
        printf( "Initialization..." );
        Initialize();
        printf( " Done\n" );

        PrepareRealTime();

        printf( "RealTime Loop - " );
        if( RunRealTime() )
        {
                printf("Interrupted\n");
                return;
        }
        printf( " Done without interruption\n" );
}
```