HXP

Hexapod
Motion Controller

Newport®

Software Drivers Manual

Intaller Pack Version #30002

Table of Contents

Hexapod Motion Controller HXP

1.0 C / C++

1.1 Using C / C++ with HXP DLL

To create a C/C++ program for the HXP, the following files are necessary:

"hxp_drivers.h" : Header file to declare function prototypes

"hxp_drivers.lib" : HXP driver library

"hxp_drivers.dll" : HXP dynamic link library

They are located in the /Admin/Public/Drivers/DLL folder of the HXP controller.

The two first files "hxp_drivers.h" and "hxp_drivers.lib" have to be added to your C++ project. (For Microsoft Visual C++, include *hxp_drivers.h and hxp_drivers.lib* in the Header Files section)

The last file "**hxp_drivers.dll**" must be in the same folder as your application or in the WINDOWS directory to be able to execute your program.

Please refer to the HXP Programmer's Manual for the descriptions of the DLL function prototypes and detailed description of what the function does. Please also read section 1.2 about memory handling of string parameters.

1.2 Memory Allocation of Function Parameters

The HXP supposes that all parameters it receives have already been allocated, this includes char*. To optimize the size, and still have a useable system, we defined four standard sizes, and char* parameters will have to be allocated accordingly with these sizes.

SIZE_SMALL = 1024

SIZE_NOMINAL = 1024

SIZE_BIG = 2048

SIZE_HUGE = 65536

By default, the size is SIZE_SMALL. Unless the function is listed here bellow:

1. SIZE_NOMINAL :
   - EventExtendedAllGet
   - GatheringDataGet
   - GroupStatusStringGet
   - HardwareInternalListGet
   - HardwareDriverAndStageGet

- PositionerDriverStatusStringGet
- PositionerErrorStringGet
- PositionerHardwareStatusStringGet

2. SIZE_BIG :
   - ActionExtendedListGet
   - ActionListGet
   - EventExtendedConfigurationTriggerGet
   - EventExtendedConfigurationActionGet
   - EventExtendedGet
   - EventGet
   - EventListGet
   - GatheringExtendedListGet
   - GatheringExternalListGet
   - GatheringListGet
   - PositionerDriverStatusListGet
   - PositionerErrorListGet
   - PositionerHardwareStatusListGet
   - ReferencingActionListGet
   - ReferencingSensorListGet
3. SIZE_HUGE :
   - APIExtendedListGet
   - APIListGet
   - ErrorListGet
   - GatheringConfigurationGet
   - GatheringExternalConfigurationGet
   - GatheringDataMultipleLinesGet
   - GroupStatusListGet
   - ObjectsListGet

Example: GatheringListGet (int socketId, char gatheringList[SIZE_BIG]);

1.3 Example of C++ Programs

1.3.1 Management of the Errors

For a safe program execution and convenient error debugging, it is recommended to check the return value of each API. One way of doing this is by using a “display error and close” program as described below. This program can be added to the project, with calls of this function after each function. In case of an error, it will indicate the name of the function at which this error occurred, the error code and the corresponding description of the error. It will also close the working socket.

Display error and close procedure in C++

```
void DisplayErrorAndClose(int error, int SocketID, char* APIName)
{
int error2;
char strError[250];

// Timeout error
if (-2 == error)
{
printf ("%s ERROR %d: TCP timeout\n", APIName, error);
TCP_CloseSocket(SocketID);
return;
}

// The TCP/IP connection was closed by an administrator
if (-108 == error)
{
printf("%s ERROR %d: The TCP/IP connection was closed by an
administrator\n",APIName,error);
return;
}

// Error => Get error description
error2 = ErrorStringGet(SocketID, error, strError);

// If error occurred with the API ErrorStringGet
if (0 != error2)
// Display API name, error code and ErrorStringGet error code
printf ("ErrorStringGet ERROR => %d\n", error2);
else
// Display API name, number and description of the error
printf ("%s ERROR => %d:  %s\n", APIName, error, strError);

// Close TCP socket
TCP_CloseSocket(SocketID);
return;
}
```

This function is called in all the following examples.

1.3.2 VersionGet

Description

This example opens a TCP connection with the HXP at the IP address specified in the variable *pIPAddress*. It gets the firmware version, print it and closes the TCP socket.

Please see the section *1.3.1 Management of the errors* for the code of the function *DisplayErrorAndClose()* function.

C++ code

```
int   error = 0;
char  buffer [SIZE_SMALL] = {'\0'};
////////////////////////////////////////////////
// TCP / IP connection
////////////////////////////////////////////////
char pIPAddress[15] = {"192.168.33.236"};
int  nPort = 5001;
double dTimeOut = 60;
int  SocketID = -1;
SocketID = TCP_ConnectToServer(pIPAddress, nPort, dTimeOut); //
Open a socket

if (-1 == SocketID)
{
printf ("Connection to @ %s, port = %ld failed\n", pIPAddress,
nPort);
return;
}
printf ("Connected to target\n");
////////////////////////////////////////////////
// Get controller version
////////////////////////////////////////////////
error = FirmwareVersionGet (SocketID, buffer); // Get controller
version
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "FirmwareVersionGet");
return;
}
else
printf ("HXP Firmware Version : %s\n", buffer);
////////////////////////////////////////////////
// TCP / IP disconnection
////////////////////////////////////////////////
TCP_CloseSocket(SocketID);  // Close Socket
printf ("Disconnected from target\n");
```

1.3.3 Gathering with Motion

Configuration

Group type	*Group name*	*Positioner name*
Hexapod	HEXAPOD	HEXAPOD.MY_STAGE

Description

This example opens a TCP connection, kills the Hexapod group, then initializes and homes it. Then, it configures the parameters for the gathering (data to be collected: setpoint and current positions). It defines an action (GatheringRun) and an event (SGamma.MotionStart). They are linked together. When the positioner moves from 0 to 50, the data is gathered (with a divisor equal to 100, data is collected every 100th servo cycle, or every 10 ms). At the end, the gathering is stopped and saved in a text file (*Gathering.dat* in /Admin/Public directory of the HXP). Finally, the program ends by closing the socket.

Please see the section *1.3.1 Management of the errors* for the code of the function *DisplayErrorAndClose()* function.

C++ code

```
int   error = 0;
char  buffer [SIZE_SMALL] = {'\0'};
/* TCP / IP connection */
char pIPAddress[15] = "192.168.33.236";
int  nPort = 5001;
double dTimeOut = 60;
int  SocketID = -1;
SocketID = TCP_ConnectToServer(pIPAddress, nPort, dTimeOut);
if (-1 == SocketID)
{
printf ("Connection to @ %s, port = %ld failed\n", pIPAddress,
nPort);
return;
}
printf ("Connected to target\n");
/* Initialization */
char* pGroup;        // Group name
char* pPositioner;        // Positioner name
char* pDataTypeList; // Types of data to be collected during the
gathering
char* pEvent;        // Event
char* pAction;       // Action triggered on the event
char* pNbPoints;          // Number of data acquisition of the
gathering
char* pDiv;               // Divisor, defining the frequency of
the gathering
char* pZero;              // Null parameter
double pDisplacement[1];  // Target displacement
int nTypes = 2;      // Number of types of data to be collected
int nAxes = 1;       // Number of axes of the group
int* eventID;        // Event ID for gathering
pGroup = "HEXAPOD";
```

```
pPositioner = " HEXAPOD.MY_STAGE"; // MY_STAGE: for exmple
1,2,3,5 or 6
pDataTypeList = " HEXAPOD.MY_STAGE.SetpointPosition
HEXAPOD.MY_STAGE.CurrentPosition";
pEvent = " HEXAPOD.MY_STAGE.SGamma.MotionStart";
pAction = "GatheringRun";
pNbPoints = "1000";
pDiv = "100";
pZero = "0";
pDisplacement[0]=50;
/* Kill group */
error = GroupKill (SocketID, pGroup);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "GroupKill");
return;
}
/* Initialize group */
error = GroupInitialize (SocketID, pGroup);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "GroupInitialize");
return;
}
/* Search home group */
error = GroupHomeSearch (SocketID, pGroup);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "GroupHomeSearch");
return;
}
/* Configure gathering */
error = GatheringConfigurationSet (SocketID, nTypes,
pDataTypeList);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID,
"GatheringConfigurationSet");
return;
}
/* Configure gathering event : trigger */
error = EventExtendedConfigurationTriggerSet
(SocketID,1,pEvent,pZero,pZero,pZero,pZero);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID,
"EventExtendedConfigurationTriggerSet");
return;
}
/* Configure gathering event : action  */
error = EventExtendedConfigurationActionSet
(SocketID,1,pAction,pNbPoints,pDiv,pZero,pZero);
if (0 != error)
{
```

```
DisplayErrorAndClose(error, SocketID,
"EventExtendedConfigurationActionSet");
return;
}
/* Configure gathering event : start */
error = EventExtendedStart (SocketID, eventID);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "EventExtendedStart");
return;
}
/* Move positioner */
error = GroupMoveRelative (SocketID, pGroup, nAxes,
pDisplacement);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "GroupMoveRelative");
return;
}
/* Stop gathering and save data */
error = GatheringStopAndSave (SocketID);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "GatheringStopAndSave");
return;
}
/* TCP / IP disconnection */
TCP_CloseSocket(SocketID);
printf ("Disconnected from target\n");
```

1.3.4 External Gathering

Configuration

Group type	*Group name*	*Positioner name*
Hexapod	HEXAPOD	HEXAPOD.MY_STAGE

Description

This example opens a TCP connection, kills the single axis group, then initializes and homes it. Then, it configures an external gathering (data to be collected: ExternalLatchPosition and GPIO2.ADC1 value). It defines an action (ExternalGatheringRun) and an event (Immediate), then link them. Each time the trigger in receives a signal; the data is gathered (with a divisor equal to 1, gathering takes place every signal on the trigger input). During gathering the current number of the gathered data gets displayed every second. At the end, the external gathering is stopped and saved in a text file (*ExternalGathering.dat* in /Admin/Public directory of the HXP). Finally, the program ends by closing the socket.

Please see the section *1.3.1 Management of the errors* for the code of the function *DisplayErrorAndClose()* function.

C++ code

```
int   error = 0;
char pIPAddress[15] = "192.168.33.236";
int  nPort = 5001;
double dTimeOut = 60;
int  SocketID = -1;
/* TCP / IP connection */
SocketID = TCP_ConnectToServer(pIPAddress, nPort, dTimeOut); //
Open a socket
if (-1 == SocketID)
{
   printf (buffer, "Connection to @ %s, port = %ld failed\n",
pIPAddress, nPort);
   return;
}
printf ("Connected to target\n");
/* Initialization */
char* pGroup;        // Group name
char* pPositioner;        // Positioner name
char* pDataTypeList; // Types of data to be collected during the
gathering
char* pEvent;        // Event name
char* pAction;       // Action triggered on the event
char* pNbPoints;          // Number of data acquisition of the
gathering in char*
int nNbPoints;       // Number of data acquisition of the
gathering in int
char* pDiv;               /* Divisor, defining every Nth number
of trigger input
 * signal at which the gathering will take place
 */
char* pZero;              // Null parameter
int nTypes = 2;      // Number of types of data to be collected
```

```
int pCurrent[1];          // Number of current acquired data
point
int pCurrentPrevious[1];  // Number of previous current acquired
data point
int pMax[1];              // Number of maximum data points per
type
int* eventID;        // Event ID for gathering
CString strResults;       // Variable for display
pGroup = "HEXAPOD";
pPositioner = " HEXAPOD.MY_STAGE"; // MY_STAGE: for exmple
1,2,3,5 or 6
pDataTypeList = " HEXAPOD.MY_STAGE.ExternalLatchPosition
GPIO2.ADC1";
pEvent = "Immediate";
pAction = "ExternalGatheringRun";
pNbPoints = "20";
nNbPoints = atoi(pNbPoints);
pDiv = "1";
pZero = "0";
pCurrent[0] = 0;
pCurrentPrevious[0] = 0;
pMax[0] = 0;
/* Kill group */
error = GroupKill (SocketID, pGroup);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "GroupKill");
return;
}
/* Initialize group */
error = GroupInitialize (SocketID, pGroup);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "GroupInitialize");
return;
}
/* Search home group */
error = GroupHomeSearch (SocketID, pGroup);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "GroupHomeSeach");
return;
}
/* Configure external gathering */
error = GatheringExternalConfigurationSet (SocketID, nTypes,
pDataTypeList);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID,
"GatheringExternalConfigurationSet");
return;
}
/* Configure gathering event : trigger */
error = EventExtendedConfigurationTriggerSet
(SocketID,1,pEvent,pZero,pZero,pZero,pZero);
```

```
if (0 != error)
{
DisplayErrorAndClose(error, SocketID,
"EventExtendedConfigurationTriggerSet");
return;
}
/* Configure gathering event : action */
error = EventExtendedConfigurationActionSet
(SocketID,1,pAction,pNbPoints,pDiv,pZero,pZero);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID,
"EventExtendedConfigurationActionSet");
return;
}
/* Configure gathering event : start */
error = EventExtendedStart (SocketID, eventID);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "EventExtendedStart");
return;
}
////////////////////////////////////////////////
// Push on TRIG ON button...
// And wait end of external gathering
////////////////////////////////////////////////
while (pCurrent[0] < nNbPoints)
{
/* Get current number realized and display it */
error = GatheringExternalCurrentNumberGet (SocketID, pCurrent,
pMax);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID,
"GatheringExternalCurrentNumberGet");
return;
}
else
{
if (pCurrentPrevious[0] < pCurrent[0])
printf ("Current gathered point : %d\n",pCurrent[0]);
     else
pCurrentPrevious[0] = pCurrent[0];
}
Sleep(1000);
}
/* Stop external gathering and save data */
error = GatheringExternalStopAndSave (SocketID);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID,
"GatheringExternalStopAndSave");
return;
}
```

```
/* TCP / IP disconnection */
TCP_CloseSocket(SocketID);
printf ("Disconnected from target\n");
```

1.3.5 Master-slave mode

Configuration

Group type	*Number*	*Group name*	*Positioner name*
Single axis	1	SINGLE_AXIS	SINGLE_AXIS.MY_STAGE
Hexapod	1	Hexapod	Hexapos.1, Hexapod.2, Hexapod.3, Hexapod.4, Hexapod.5 and Hexapod.6

Description

This example opens a TCP connection, kills the singles axis and XY group, then initializes and homes them. It sets the parameters for the master slave mode (slave: single axis group, master: X positioner from XY group). Then, it enables the master slave mode and executes a relative move of 65 units on the master positioner. Simultaneously, the slave positioner executes the same move as the master. The master slave mode is then disabled and the program ends by closing the socket.

Please see the section *1.3.1 Management of the errors* for the code of the function *DisplayErrorAndClose()* function.

C++ code

```
   int   error = 0;
   /* TCP / IP connection */
   char pIPAddress[15] = "192.168.33.236";
   int  nPort = 5001;
   double dTimeOut = 60;
   int  SocketID = -1;
   SocketID = TCP_ConnectToServer(pIPAddress, nPort, dTimeOut);
// Open a socket
   if (-1 == SocketID)
   {
printf ("Connection to @ %s, port = %ld failed\n", pIPAddress,
nPort);
return;
}
   printf ("Connected to target\n");
   /* Initialization */
   char* pSlaveGroup;      // Slave single axis group name
   char* pXYGroup;         // XY group name
   char* pMasterPositioner;     // Master positioner name
   double pDisplacement[1];     // Target displacement
   int nAxes = 1;          // Number of axes of the group
   double dMasterRatio = 1;     // Ratio defining the slave copy:
Slave = ratio * Master
   pSlaveGroup = "SINGLE_AXIS";
   pXYGroup = "XY";
   pMasterPositioner = "XY.X";
   pDisplacement[0]=65;
   /* Kill single axis group */
   error = GroupKill (SocketID, pSlaveGroup);
   if (0 != error)
   {
        DisplayErrorAndClose(error, SocketID, "Single axis
GroupKill");
```

```
        return;
    }
    /* Initialize single axis group */
    error = GroupInitialize (SocketID, pSlaveGroup);
    if (0 != error)
    {
        DisplayErrorAndClose(error, SocketID, "Single axis
GroupInitialize");
        return;
    }
    /* Search home single axis group */
    error = GroupHomeSearch (SocketID, pSlaveGroup);
    if (0 != error)
    {
        DisplayErrorAndClose(error, SocketID, "Single axis
GroupHomeSearch");
        return;
    }
    /* Kill XY group */
    error = GroupKill (SocketID, pXYGroup);
    if (0 != error)
    {
        DisplayErrorAndClose(error, SocketID, "XY GroupKill");
        return;
    }
    /* Initialize XY group */
    error = GroupInitialize (SocketID, pXYGroup);
    if (0 != error)
    {
        DisplayErrorAndClose(error, SocketID, "XY
GroupInitialize");
        return;
    }
    /* Search home XY group */
    error = GroupHomeSearch (SocketID, pXYGroup);
    if (0 != error)
    {
        DisplayErrorAndClose(error, SocketID, "XY
GroupHomeSearch");
        return;
    }
    /* Set slave (single axis group) with its master (positioner
from any group) */
    error = SingleAxisSlaveParametersSet (SocketID, pSlaveGroup,
pMasterPositioner, dMasterRatio);
    if (0 != error)
    {
        DisplayErrorAndClose(error, SocketID,
"SingleAxisSlaveParametersSet");
        return;
    }
    /* Enable slave-master mode */
    error = SingleAxisSlaveModeEnable (SocketID, pSlaveGroup);
    if (0 != error)
    {
```

```
        DisplayErrorAndClose(error, SocketID,
"SingleAxisSlaveModeEnable");
        return;
   }
   /* Move master positioner (the slave must follow the master in
relation to the ratio) */
   error = GroupMoveRelative (SocketID, pMasterPositioner, nAxes,
pDisplacement);
   if (0 != error)
   {
DisplayErrorAndClose(error, SocketID, "GroupMoveRelative");
return;
   }
   /* Disable slave-master mode */
   error = SingleAxisSlaveModeDisable (SocketID, pSlaveGroup);
   if (0 != error)
   {
        DisplayErrorAndClose(error, SocketID,
"SingleAxisSlaveModeDisable");
        return;
   }
   /* TCP / IP disconnection */
   TCP_CloseSocket(SocketID);
   printf ("Disconnected from target\n");
```

1.3.6 Backlash Compensation

Configuration

Group type	*Group name*	*Positioner name*
Hexapod	HEXAPOD	HEXAPOD.MY_STAGE

Description

This example opens a TCP connection and kills the Hexapod group. It enables the backlash compensation capability (for this the group must be in the not_initialized state). The group gets initialized and homed. The value of the backlash compensation is set to 0.1. The positioner executes relative moves with the backlash compensation. Finally, the backlash compensation gets disabled and the program ends by closing the socket.

CAUTION

- **The *HomeSearchSequenceType* in the *stages.ini* file must be different than *CurrentPositionAsHome*.**
- ***The Backlash* parameter in the *stages.ini* file must be greater than zero.**
- **To apply any modifications of the *stages.ini*, the controller must be rebooted.**

Please see the section *1.3.1 Management of the errors* for the code of the function *DisplayErrorAndClose()* function.

C++ code

```
int   error = 0;
/* TCP / IP connection */
char pIPAddress[15] = "192.168.33.236";
int  nPort = 5001;
double dTimeOut = 60;
int  SocketID = -1;
SocketID = TCP_ConnectToServer(pIPAddress, nPort, dTimeOut); //
Open a socket
if (-1 == SocketID)
{
printf ("Connection to @ %s, port = %ld failed\n", pIPAddress,
nPort);
return;
}
printf("Connected to target\n");
/* Initialization */
char* pGroup;         // Group name
char* pPositioner;          // Positioner name
double pDisplacement1[1]; // Target positive displacement
double pDisplacement2[1]; // Target negative displacement
double dBacklash = 0.1;    // New Backlash value
int nAxes = 1;        // Nuber of axes of the group
pGroup = "HEXAPOD";
pPositioner = " HEXAPOD.MY_STAGE"; // MY_STAGE: for exmple
1,2,3,5 or 6
pDisplacement1[0] = 10;
```

```
pDisplacement2[0] = -10;
/* Kill group */
error = GroupKill (SocketID, pGroup);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "GroupKill");
return;
}
/* Enable Backlash
 * Caution : Group must be "Not_Initialized" and Backlash > 0 in
"stages.ini"
 */
error = PositionerBacklashEnable(SocketID, pPositioner);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID,
"PositionerBacklashEnable");
return;
}
/* Initialize group */
error = GroupInitialize (SocketID, pGroup);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "GroupInitialize");
return;
}
/* Search home group */
error = GroupHomeSearch (SocketID, pGroup);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "GroupHomeSearch");
return;
}
/* Modify Backlash value. Caution : Backlash > 0 in "stages.ini"
*/
error = PositionerBacklashSet (SocketID, pPositioner, dBacklash);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "PositionerBacklashSet");
return;
}
/* Move in positive direction */
error = GroupMoveRelative (SocketID, pGroup, nAxes,
pDisplacement1);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "GroupMoveRelative");
return;
}
/* Move in negative direction */
error = GroupMoveRelative (SocketID, pGroup, nAxes,
pDisplacement2);
if (0 != error)
{
```

```
DisplayErrorAndClose(error, SocketID, "GroupMoveRelative");
return;
}
/* Disable Backlash */
error = PositionerBacklashDisable (SocketID, pPositioner);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID,
"PositionerBacklashDisable");
return;
}
/* TCP / IP disconnection */
TCP_CloseSocket(SocketID);
printf ("Disconnected from target\n");
```

1.3.7 Timer Event and Global Variables

Configuration

Group type	*Group name*	*Positioner name*
Hexapod	HEXAPOD	HEXAPOD.MY_STAGE

Description

The main program opens a TCP connection, configures a timer and uses this timer as an event. The action, in relation to this timer event, executes a second TCL script named *MyScript.tcl*. The main program sets a global variable and closes the socket.

The timer is a permanent event. The frequency of the timer is set by the divisor, in this example 20000, which means that the second TCL script gets executed every 20000th servo loop or every 2 seconds (divisor/servo loop rate = 20000/10000 = 2 seconds).

The script *MyScript.tcl* reads the global variable, increments it as long as the variable is below 10. When the global variable is equal to 10, the second script deletes the timer event and finally, the program ends by closing the socket.

Please see the section *1.3.1 Management of the errors* for the code of the function *DisplayErrorAndClose()* function.

C++ code

```
int   error = 0;
/* TCP / IP connection */
char pIPAddress[15] = "192.168.33.236";
int  nPort = 5001;
double dTimeOut = 60;
int  SocketID = -1;
SocketID = TCP_ConnectToServer(pIPAddress, nPort, dTimeOut); //
Open a socket
if (-1 == SocketID)
{
printf ("Connection to @ %s, port = %ld failed\n", pIPAddress,
nPort);
return;
}
printf ("Connected to target\n");
/* Initialization */
char* pPositioner;              // Positioner name
char* pTimer;             // Timer name
char* pEvent;             // Event name
char* pZero;                    // Null parameter
char* pAction;            // Action triggered on the event
char* pTCLFile;           // Name of the TCL script to be
executed
char* pTCLTask;           // Name of the TCL task
char* pTCLArgs;           // Argument list of the TCL task
char* pValue;             // Value of the global variable
double dISRPeriodSec = 0.0001;  // Value of ISR period
double dTimerPeriodSec = 2;          // Value of timer period
int nDivisor = 0;               // Frequency ticks of the timer
int nGlobalVarNumber = 1;       // Number of global variable
int* eventID;             // Event ID
```

```
CString strResults;             // Variable for display
pPositioner = "HEXAPOD.MY_STAGE";  // MY_STAGE: for exmple
1,2,3,5 or 6
pTimer = pEvent = "Timer1";
pZero = "0";
pAction = "ExecuteTCLScript";
pTCLFile = "MyScript.tcl";
pTCLTask = "MyTask";
pTCLArgs = "0";
pValue = "5";
/* Calculate divisor */
nDivisor = (int) (dTimerPeriodSec / dISRPeriodSec);
printf ("Divisor value: %d\n",nDivisor);
/* Configure Timer */
error = TimerSet (SocketID, pTimer, nDivisor);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "TimerSet");
return;
}
/* Configure timer event : trigger  */
error = EventExtendedConfigurationTriggerSet
(SocketID,1,pEvent,pZero,pZero,pZero,pZero);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID,
"EventExtendedConfigurationTriggerSet");
return;
}
/* Configure tcl execute action  */
error = EventExtendedConfigurationActionSet
(SocketID,1,pAction,pTCLFile,pTCLTask,pTCLArgs,pZero);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID,
"EventExtendedConfigurationActionSet");
return;
}
/* Start event */
error = EventExtendedStart (SocketID, eventID);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "EventExtendedStart");
return;
}
/* Set global variable */
error = GlobalArraySet  (SocketID, nGlobalVarNumber, pValue);
if (0 != error)
{
DisplayErrorAndClose(error, SocketID, "GlobalArraySet");
return;
}

/* TCP / IP disconnection */
```

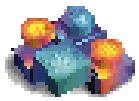
```
TCP_CloseSocket(SocketID);
printf ("Disconnected from target\n");
```

MyScript.tcl

```
# Display error and close procedure
proc DisplayErrorAndClose {socketID code APIName} {
global tcl_argv
if {$code != -2 && $code != -108} {
     set code2 [catch "ErrorStringGet $socketID $code strError"]
     if {$code2 != 0} {
          puts stdout "$APIName ERROR => $code - ErrorStringGet
ERROR => $code2"
          set tcl_argv(0) "$APIName ERROR => $code"
     } else {
          puts stdout "$APIName $strError"
          set tcl_argv(0) "$APIName $strError"
     }
} else {
     if {$code == -2} {
          puts stdout "$APIName ERROR => $code : TCP timeout"
          set tcl_argv(0) "$APIName ERROR => $code : TCP timeout"
     }
     if {$code == -108} {
          puts stdout "$APIName ERROR => $code : The TCP/IP
connection was closed by an administrator"
          set tcl_argv(0) "$APIName ERROR => $code :  The TCP/IP
connection was closed by an administrator"
     }
set code2 [catch "TCP_CloseSocket $socketID"]
return
}
# Main process
set TCPTimeOut 0.5
set code 0
set GlobalVarNumber 1
set ReadValue 0
set NewValue 0
set END 10
set Positioner "SINGLE_AXIS.MY_STAGE"
set EventName "Timer1"
set EventPara 0
# open TCP socket
OpenConnection $TCPTimeOut SocketID
if {$socketID == -1} {
puts stdout "OpenConnection failed => $socketID"
return
}
# Read global variable
set code [catch "GlobalArrayGet $socketID $GlobalVarNumber
ReadValue"]
if {$code != 0} {
DisplayErrorAndClose $socketID $code "GlobalArrayGet"
return
}
```

```
if { $ReadValue < $END } {
# Increment global variable
set NewValue [expr {$ReadValue + 1}]
# Set global variable
set code [catch "GlobalArraySet $socketID $GlobalVarNumber
$NewValue"]
if {$code != 0} {
DisplayErrorAndClose $socketID $code "GlobalArraySet"
return
} else {
puts stdout "New value: $NewValue"
}
} else {
# Delete timer event
set code [catch "EventRemove $socketID $Positioner $EventName
$EventPara"]
if {$code != 0} {
DisplayErrorAndClose $socketID $code "GlobalArraySet"
return
} else {
puts "Timer event deleted"
}
}
# close TCP socket
set code [catch "TCP_CloseSocket $socketID"]
```

2.0 Visual Basic 6 Drivers



2.1 Visual Basic 6 with HXP DLL

To write a program for the HXP with Visual Basic 6, you must add the file "hxp_drivers.bas" to your Visual Basic project. The file "hxp_drivers.bas" contains function declarations from the HXP drivers. It uses the HXP DLL. Therefore, the needed files are:

"hxp_drivers.bas" : Visual basic interface file to declare function prototypes from HXP driver

"hxp_drivers.lib" : HXP driver library

"hxp_drivers.dll" : HXP dynamic link library

They are located in the ../Admin/Public/Drivers/dll directory of the HXP controller.

The "**hxp_drivers.bas**" file (Visual basic) is built in relation to "hxp_drivers.h" (Visual C++).

In Visual Basic, each API prototype is described as follow:

Declare **Function** *API name* **Lib** "*FileName.dll*" (*API Parameters*) **As** *ReturnedType*
Declare **Sub** *API name* **Lib** "*FileName.dll*" (*API Parameters*)

Example:

Visual Basic (.BAS):

```
Declare Function TCP_ConnectToServer Lib "hxp_drivers.dll" (ByVal
Ip_Address As String, ByVal Ip_Port As Integer, ByVal TimeOut As
Double) As Long
```

To execute your Visual Basic application, the file "**hxp_drivers.dll**" must be in the WINDOWS directory or in the current directory of the executed program. Please refer to the HXP Programmer's Manual for the descriptions of the function prototypes and detailed description of what the function does.

2.2 Issue with Boolean

Visual Basic 6 has by default a value of 0 for False and –1 for True. In our system, because we use the same dll for C/C++, True is by default 1. This is an issue we can not handle as it comes from Microsoft definition of their language.

This should not affect functions that read a Boolean value from our controller, as True is actually defined as "all value different than 0". But for functions that write a Boolean (for example : PositionerCorrectorPIDFFAccelerationSet), it will cause problems. Therefore, we advise to always use the absolute value of your Boolean when you send a Boolean to our system (it should look like "Abs(MyBoolean)").

2.3 Example of a Visual Basic program

```
Public Buffer As String
Public IPAddress As String
Public IPPort As Integer
Public SocketID As Integer
Private Sub Form_Load()
    SocketID = -1
    IPAddress = "192.168.33.234"
    IPPort = 5001
    Buffer = String(512 + 1, 0)
End Sub
```

```
Private Sub Application_Click()
    Dim error As Integer
    Dim AnalogValue() As Double
    Dim AnalogNameList As String
    ReDim AnalogValue(4)
'//////////////////////////////////////
    ' Open TCP IP connection
    '//////////////////////////////////////
    SocketID = TCP_ConnectToServer(IPAddress, IPPort, 10)
    If SocketID <> -1 Then
'//////////////////////////////////////
        ' Get firmware version
        '//////////////////////////////////////
        error = FirmwareVersionGet(SocketID, ByVal Buffer)
        Message.Text = Buffer
'//////////////////////////////////////
        ' Set GPIO analog output
        '//////////////////////////////////////
        AnalogNameList =
"GPIO2.DAC1;GPIO2.DAC2;GPIO2.DAC3;GPIO2.DAC4"
        AnalogValue(0) = 1
        AnalogValue(1) = 2
        AnalogValue(2) = 3
        AnalogValue(3) = 4
error = GPIOAnalogSet(SocketID, 4, AnalogNameList,
AnalogValue(0))
        If (error = 0) Then
'//////////////////////////////////////
            ' Get GPIO analog output
            '//////////////////////////////////////
            error = GPIOAnalogGet(SocketID, 4, AnalogNameList,
AnalogValue(0))
            If (error = 0) Then
                Message.Text = "DAC1 = " & AnalogValue(0) & "
DAC2 = "&AnalogValue(1) & " DAC3 =
                                "&AnalogValue(2) & " DAC4 =
"&AnalogValue(3)
            End If
        End If
'//////////////////////////////////////
        ' Get error
        '//////////////////////////////////////
        If error <> 0 Then
            error = ErrorStringGet(SocketID, error, ByVal Buffer)
            Message.Text = Buffer
        End If
'//////////////////////////////////////
        ' Close TCP IP connection
        '//////////////////////////////////////
        TCP_CloseSocket (SocketID)
        SocketID = -1
End If
```

3.0 Matlab Drivers

3.1 HXP with Matlab

At first, the HXP APIs library and m files have to be unzipped into a folder, and set into Matlab path. To do so, use your usual unzipper anywhere you want. Then in Matlab, click on the menu “File”, submenu “Set path…”. Click on “Add folder” button, and browse to your unzipped folder. Click on “OK”, then “Save” and “Close”. You are now able to use the HXP library for Matlab

First, you have to load the library into Matlab memory. This is done using the following function : “`hxp_load_drivers`”. Calling this function more than one time won’t cause any issue. You will just be warned you already did it before.

Now you can call a function with : `[returnedValue1, returnedValue2, …] = API (parameter1, parameter2, …)`

3.2 Help

When using a new function, you may want to use the help function of Matlab. A short comment about the function, and the complete prototype will be given to you.

Example:

```
>> help EventExtendedConfigurationActionGet
        EventExtendedConfigurationActionGet :  Read the action
configuration
[errorCode, ActionConfiguration] =
EventExtendedConfigurationActionGet(socketId)
* Input parameters :
          int32 socketId
     * Output parameters :
          int32 errorCode
          cstring ActionConfiguration
```

To have more information about any function prototype, or what the action does, see the Programmer’s manual.

3.2.1 Example of a Matlab Program

```
% Load the library
hxp_load_drivers ;
% Set connection parameters
IP = '192.168.33.234' ;
Port = 5001 ;
TimeOut = 60.0 ;
% Connect to HXP
socketID = TCP_ConnectToServer (IP, Port, TimeOut) ;
% Check connection
if (socketID < 0)
    disp 'Connection to HXP failed, check IP & Port' ;
    return ;
end
% Define the positioner
group = 'HEXAPOD' ;
positioner = ' HEXAPOD.POSITIONER' ;
% Kill the group
[errorCode] = GroupKill(socketID, group) ;
if (errorCode ~= 0)
```

```
    disp (['Error ' num2str(errorCode) ' occurred while doing
GroupKill ! ']) ;
    return ;
end
% Initialize the group
[errorCode] = GroupInitialize(socketID, group) ;
if (errorCode ~= 0)
    disp (['Error ' num2str(errorCode) ' occurred while doing
GroupInitialize ! ']) ;
    return ;
end
% Home search
[errorCode] = GroupHomeSearch(socketID, group) ;
if (errorCode ~= 0)
    disp (['Error ' num2str(errorCode) ' occurred while doing
GroupHomeSearch ! ']) ;
    return ;
end
% Make a move
[errorCode] = GroupMoveAbsolute(socketID, positioner, 20.0) ;
if (errorCode ~= 0)
    disp (['Error ' num2str(errorCode) ' occurred while doing
GroupMoveAbsolute ! ']) ;
    return ;
end
% Get current position
[errorCode, currentPosition] = GroupPositionCurrentGet(socketID,
positioner, 1) ;
if (errorCode ~= 0)
    disp (['Error ' num2str(errorCode) ' occurred while doing
GroupPositionCurrentGet! ']) ;
    return ;
else
    disp (['Positioner ' positioner ' is in position '
num2str(currentPosition)]) ;
end
% Close connection
TCP_CloseSocket(socketID) ;
```

4.0 Python Drivers

4.1 HXP with Python

The Python interface to the HXP comes in a file '*hxp_drivers.py*' that describes a class *HXP*, with all the HXP functions declared in it. You will need to have it in the same directory as your Python program. It is located on the HXP in the directory /Admin/Public/Drivers/python.

To use this class, you need to import it in your program. This is done in the following way:

```
import hxp_drivers
```

If you need more information about a function prototype, or what the action does, see the Programmer's manual.

4.2 Example of Python Program

```
# --------- Python program : HXP controller demonstration -------
- #
import hxp_drivers
import sys
# Display error function : simplify error print out and closes
socket
def displayErrorAndClose (socketId, errorCode, APIName):
if (errorCode != -2) and (errorCode != -108):
     [errorCode2, errorString] = myxps.ErrorStringGet(socketId,
errorCode)
     if (errorCode2 != 0):
          print APIName + ' : ERROR ' + str(errorCode)
     else:
          print APIName + ' : ' + errorString
else:
     if (errorCode == -2):
          print APIName + ' : TCP timeout'
     if (errorCode == -108):
          print APIName + ' : The TCP/IP connection was closed by
an administrator'
myxps.TCP_CloseSocket(socketId)
return
# Instantiate the class
myxps = hxp_drivers.XPS()
# Connect to the HXP
socketId = myxps.TCP_ConnectToServer('192.168.33.235', 5001, 20)
# Check connection passed
if (socketId == -1):
print 'Connection to HXP failed, check IP & Port'
sys.exit ()
# Add here your personal codes, below for example :
# Define the positioner
group = 'HEXAPOD'
positioner = group + '.1'
# Kill the group
[errorCode, returnString] = myxps.GroupKill(socketId, group)
if (errorCode != 0):
displayErrorAndClose (socketId, errorCode, 'GroupKill')
```

```
sys.exit ()
# Initialize the group
[errorCode, returnString] = myxps.GroupInitialize(socketId,
group)
if (errorCode != 0):
displayErrorAndClose (socketId, errorCode, 'GroupInitialize')
sys.exit ()
# Home search
[errorCode, returnString] = myxps.GroupHomeSearch(socketId,
group)
if (errorCode != 0):
displayErrorAndClose (socketId, errorCode, 'GroupHomeSearch')
exit
# Make some moves
for index in range(10):
# Forward
[errorCode, returnString] = myxps.GroupMoveAbsolute(socketId,
positioner, [20.0])
if (errorCode != 0):
     displayErrorAndClose (socketId, errorCode,
'GroupMoveAbsolute')
     sys.exit ()
# Get current position
[errorCode, currentPosition] =
myxps.GroupPositionCurrentGet(socketId, positioner, 1)
if (errorCode != 0):
     displayErrorAndClose (socketId, errorCode,
'GroupPositionCurrentGet')
     sys.exit ()
else:
     print 'Positioner ' + positioner + ' is in position ' +
str(currentPosition)
# Backward
[errorCode, returnString] = myxps.GroupMoveAbsolute(socketId,
positioner, [-20.0])
if (errorCode != 0):
     displayErrorAndClose (socketId, errorCode,
'GroupMoveAbsolute')
     sys.exit ()
# Get current position
[errorCode, currentPosition] =
myxps.GroupPositionCurrentGet(socketId, positioner, 1)
if (errorCode != 0):
     displayErrorAndClose (socketId, errorCode,
'GroupPositionCurrentGet')
     sys.exit ()
else:
     print 'Positioner ' + positioner + ' is in position ' +
str(currentPosition)
# Close connection
myxps.TCP_CloseSocket(socketId)
#---------- End of the demo program ----------#
```

Service Form

Your Local Representative

Tel.: _____

Fax: _____

Name: _____

Company: _____

Address: _____

Country: _____

P.O. Number: _____

Item(s) Being Returned: _____

Model#: _____

Return authorization #: _____
(Please obtain prior to return of item)

Date: _____

Phone Number: _____

Fax Number: _____

Serial #: _____

Description: _____

Reasons of return of goods (please list any specific problems): _____

Newport®

Visit Newport Online at:
www.newport.com

North America & Asia
Newport Corporation
1791 Deere Ave.
Irvine, CA 92606, USA

Sales
Tel.: (800) 222-6440
e-mail: sales@newport.com

Technical Support
Tel.: (800) 222-6440
e-mail: tech@newport.com

Service, RMAs & Returns
Tel.: (800) 222-6440
e-mail: service@newport.com

Europe
MICRO-CONTROLE Spectra-Physics S.A.S
9, rue du Bois Sauvage
91055 Évry CEDEX
France

Sales
Tel.: +33 (0)1.60.91.68.68
e-mail: france@newport.com

Technical Support
e-mail: tech_europe@newport.com

Service & Returns
Tel.: +33 (0)2.38.40.51.55

mks

Newport® Ophir® Spectra-Physics®