ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Newport XPS-C8* at our website: ***Click HERE***

XPS Controller

Universal High-Performance Motion Controller/Driver



Software drivers manual
V2.5.x

Spectra-Physics
A Division of Newport Corporation

Newport®
Experience | Solutions

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Confidentiality & Proprietary Rights

Reservation of Title :

The Newport Programs and all materials furnished or produced in connection with them ("Related Materials") contain trade secrets of Newport and are for use only in the manner expressly permitted. Newport claims and reserves all rights and benefits afforded under law in the Programs provided by Newport Corporation.
Newport shall retain full ownership of Intellectual Property Rights in and to all development, process, align or assembly technologies developed and other derivative work that may be developed by Newport. Customer shall not challenge, or cause any third party to challenge, the rights of Newport.

Preservation of Secrecy and Confidentiality and Restrictions to Access:

Customer shall protect the Newport Programs and Related Materials as trade secrets of Newport, and shall devote its best efforts to ensure that all its personnel protect the Newport Programs as trade secrets of Newport Corporation. Customer shall not at any time disclose Newport's trade secrets to any other person, firm, organization, or employee that does not need (consistent with Customer's right of use hereunder) to obtain access to the Newport Programs and Related Materials. These restrictions shall not apply to information (1) generally known to the public or obtainable from public sources; (2) readily apparent from the keyboard operations, visual display, or output reports of the Programs; (3) previously in the possession of Customer or subsequently developed or acquired without reliance on the Newport Programs; or (4) approved by Newport for release without restriction.

2003 Newport Corporation
1791 Deere Ave.
Irvine, CA 92606, USA
(949) 863-3144

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table of Contents

1 C / C++ 4

1.1 Using C / C++ with XPS DLL 4
1.2 Memory allocation of function parameters 5
1.3 Example of C++ programs 6
1.3.1 Management of the errors 6
1.3.2 VersionGet 7
1.3.3 Gathering with motion 8
1.3.4 External gathering 10
1.3.5 Position Compare 13
1.3.6 Master-slave mode 15
1.3.7 Jogging 17
1.3.8 Analog position tracking 19
1.3.9 Backlash compensation 21
1.3.10 Timer event and global variables 23
1.3.11 Pulse generation on trajectory (with gathering) 26

2 Visual Basic 6 Drivers 29

2.1 Visual Basic 6 with XPS DLL 29
2.2 Issue with Boolean 29
2.3 Example of a Visual Basic program 30

3 Matlab Drivers 31

3.1 XPS with Matlab 31
3.2 Help 31
3.3 Example of a Matlab program 32

4 Python Drivers 33

4.1 XPS with Python 33
4.2 Example of Python program 34

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



1 C / C++

1.1 Using C / C++ with XPS DLL

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

"XPS_C8_drivers.h" : Header file to declare function prototypes
"XPS_C8_drivers.lib" : XPS-C8 driver library
"XPS_C8_drivers.dll" : XPS-C8 dynamic link library

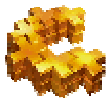
They are located in the /Admin/Public/Drivers/DLL folder of the XPS controller.

The two first files "XPS_C8_drivers.h" and "XPS_C8_drivers.lib" have to be added to your C++ project. (For Microsoft Visual C++, include *XPS_C8_drivers.h and XPS_C8_drivers.lib* in the Header Files section)

The last file "**XPS_C8_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 XPS 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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



1.2 Memory allocation of function parameters

From XPS Firmware Version V4.2.0 Precision Platform, memory handling has been improved to avoid over-use of memory on the stack. Now big buffers are allocated on the heap. The XPS 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
   - PositionerWarningStringGet

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]);`

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



1.3.2 VersionGet

Description: This example opens a TCP connection with the XPS 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 ("XPS Firmware Version : %s\n", buffer);

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



1.3.3 Gathering with motion

Configuration:

Group type	*Number*	*Group name*	*Positioner name*
Single axis	1	SINGLE_AXIS	SINGLE_AXIS.MY_STAGE

Description: This example opens a TCP connection, kills the single axis 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 XPS). 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 = "SINGLE_AXIS";
pPositioner = "SINGLE_AXIS.MY_STAGE";
pDataTypeList = "SINGLE_AXIS.MY_STAGE.SetpointPosition SINGLE_AXIS.MY_STAGE.CurrentPosition";
pEvent = "SINGLE_AXIS.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);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



```
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");
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



1.3.4 External gathering

Configuration:

Group type	*Number*	*Group name*	*Positioner name*
Single axis	1	SINGLE_AXIS	SINGLE_AXIS.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 XPS). 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 = "SINGLE_AXIS";
pPositioner = "SINGLE_AXIS.MY_STAGE";
pDataTypeList = "SINGLE_AXIS.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;
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



```
/* 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
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
                    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");
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



1.3.5 Position Compare

Configuration:

Group type	*Number*	*Group name*	*Positioner name*
Single axis	1	SINGLE_AXIS	SINGLE_AXIS.MY_STAGE

Description: This example opens a TCP connection, kills the single axis group, then initializes and homes it. With an absolute move, the positioner moves to the start position –15. Then, it configures the parameters for the position compare (enabled from –10 to +10 with step position of 1 unit). It enables the position compare functionality and executes a relative move of 25 (positioner final position will be –15+25 = +10). During this move, between the positions –10 and +10, pulses are sent by the trigger output when crossing each 1 unit incremental position. The position compare 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* pGroup;              // Group name
char* pPositioner;         // Positioner name
double pStartPosition[1];  // Diplacement to start position
double pDisplacement[1];   // Target diplacement
double dMinPos = -10;      // Minimum position for which the output compare is activated
double dMaxPos = 10;       // Maximum position for which the output compare is activated
double dStepPos = 1;       // Step position of the pulses during the output compare
int nAxes = 1;             // Number of axes of the group

pGroup = "SINGLE_AXIS";
pPositioner = "SINGLE_AXIS.MY_STAGE";
pStartPosition[0]=-15;
pDisplacement[0]=25;

/* 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)
{
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



```
        DisplayErrorAndClose(error, SocketID, "GroupHomeSearch");
        return;
    }

    /* Move positioner to start position */
    error = GroupMoveAbsolute (SocketID, pGroup, nAxes, pStartPosition);
    if (0 != error)
    {
        DisplayErrorAndClose(error, SocketID, "GroupMoveAbsolute");
        return;
    }

    /* Set position compare parameters */
    error = PositionerPositionCompareSet (SocketID, pPositioner, dMinPos, dMaxPos, dStepPos);
    if (0 != error)
    {
        DisplayErrorAndClose(error, SocketID, "PositionerPositionCompareSet");
        return;
    }

    /* Enable position compare mode */
    error = PositionerPositionCompareEnable (SocketID, pPositioner);
    if (0 != error)
    {
        DisplayErrorAndClose(error, SocketID, "PositionerPositionCompareEnable");
        return;
    }

    /* Move positioner */
    error = GroupMoveAbsolute (SocketID, pGroup, nAxes, pDisplacement);
    if (0 != error)
    {
        DisplayErrorAndClose(error, SocketID, "GroupMoveAbsolute");
        return;
    }

    /* Disable position compare mode */
    error = PositionerPositionCompareDisable (SocketID, pPositioner);
    if (0 != error)
    {
        DisplayErrorAndClose(error, SocketID, "PositionerPositionCompareDisable");
        return;
    }

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



1.3.6 Master-slave mode

Configuration:

Group type	*Number*	*Group name*	*Positioner name*
Single axis	1	SINGLE_AXIS	SINGLE_AXIS.MY_STAGE
XY	1	XY	XY.X and XY.Y

Description: This example opens a TCP connection, kills the singles axis and the 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 with 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;
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



```
}

/* 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");
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

1.3.7 Jogging

Configuration:

Group type	*Number*	*Group name*	*Positioner name*
XY	1	XY	XY.X and XY.Y

Description: This example opens a TCP connection, kills the XY group, then initializes and homes it. It enables the jog mode and sets the parameters to move a positioner in the positive direction with a velocity of 20 units/s for 3 seconds. Then, during the next 3 seconds, the positioner moves in the reverse direction with a velocity of -30 units/s, and finally stops (velocity set to 0). The jog functionality 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* pGroup;              // Group name
char* pPositioner;         // Positioner name
int nPositioners = 2;      // Number of positioners in the group
double pVelocity1[2];      // Velocity demanded during first jog operation of the pos.
double pVelocity2[2];      // Velocity demanded during second jog operation of the pos.
double pNullVelocity[2];   // Null velocity demanded during third jog operation of the pos.
double pAcceleration[2];   // Acceleration demanded during jog operations of the positioner

pGroup = "XY";
pPositioner = "XY.X";
pVelocity1[0] = 20;        // Velocity of the X positioner demanded during first jog op
pVelocity1[1] = 20;        // Velocity of the Y positioner demanded during first jog op
pVelocity2[0] = -30;       // Velocity of the X positioner demanded during second jog op
pVelocity2[1] = -30;       // Velocity of the Y positioner demanded during second jog op
pNullVelocity[0] = 0;      // Null velocity of the X pos. demanded during third jog op
pNullVelocity[1] = 0;      // Null velocity of the Y pos. demanded during third jog op
pAcceleration[0] = 80;     // Acceleration of the X positioner demanded during jog op
pAcceleration[1] = 80;     // Acceleration of the Y positioner demanded during jog op

/* 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;
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



```
/* Search home group */
error = GroupHomeSearch (SocketID, pGroup);
if (0 != error)
{
      DisplayErrorAndClose(error, SocketID, "GroupHomeSearch");
      return;
}

/* Enable jog mode */
error = GroupJogModeEnable (SocketID, pGroup);
if (0 != error)
{
      DisplayErrorAndClose(error, SocketID, "GroupJogModeEnable");
      return;
}

/* Set jog parameters to move a positioner => constant velocity is not null */
error = GroupJogParametersSet (SocketID, pGroup, nPositioners, pVelocity1, pAcceleration);
if (0 != error)
{
      DisplayErrorAndClose(error, SocketID, "GroupJogParametersSet");
      return;
}

/* Wait 3 seconds */
Sleep(3000);

/* Set Jog parameters to move the positioner in the
 * reverse sense => constant velocity is not null
 */
error = GroupJogParametersSet (SocketID, pGroup, nPositioners, pVelocity2, pAcceleration);
if (0 != error)
{
      DisplayErrorAndClose(error, SocketID, "GroupJogParametersSet");
      return;
}

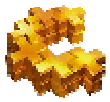
/* Wait 3 seconds */
Sleep(3000);

/* Set Jog parameters to stop a positioner => constant velocity is null */
error = GroupJogParametersSet (SocketID, pGroup, nPositioners, pNullVelocity, pAcceleration);
if (0 != error)
{
      DisplayErrorAndClose(error, SocketID, "GroupJogParametersSet");
      return;
}

/* Disable Jog mode (constant velocity must be null on all positioners from group) */
error = GroupJogModeDisable (SocketID, pGroup);
if (0 != error)
{
      DisplayErrorAndClose(error, SocketID, "GroupJogModeDisable");
      return;
}

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



1.3.8 Analog position tracking

Configuration:

Group type	*Number*	*Group name*	*Positioner name*
XY	1	XY	XY.X and XY.Y

Description: This example opens a TCP connection, kills the XY group, then initializes and homes it. It sets the parameters for the position analog tracking functionality (positioner, analog input, offset, scale, velocity and acceleration) and enables the analog tracking mode. The mode gets activated during 20 seconds. During this time, the stage follows in position the voltage of the analog input GPIO2.ADC1. Then, the analog tracking mode gets 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 (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* pType;               // Type of analog tracking (position or velocity)
char* pAnalogInput;        // Analog input controlling the tracking
double dOffset = 0;        // Offset of the positioner moves during tracking
double dScale = 1;         // Scale of the positioner moves during tracking
double dVelocity = 20;     // Velocity of the positioner during tracking
double dAcceleration = 80; // Acceleration of the positioner during tracking

pGroup = "XY";
pPositioner = "XY.X";
pType = "Position";
pAnalogInput = "GPIO2.ADC1";

/* 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");
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



```
        return;
    }

    /* Set tracking parameters */
    error = PositionerAnalogTrackingPositionParametersSet (SocketID, pPositioner, pAnalogInput,
dOffset, dScale, dVelocity, dAcceleration);
    if (0 != error)
    {
        DisplayErrorAndClose(error, SocketID,"PositionerAnalogTrackingPositionParametersSet");
        return;
    }

    /* Enable tracking mode */
    error = GroupAnalogTrackingModeEnable (SocketID, pGroup, pType);
    if (0 != error)
    {
        DisplayErrorAndClose(error, SocketID, "GroupAnalogTrackingModeEnable");
        return;
    }

    /* Change the amplitude of analog input during 20 seconds */
    Sleep(20000);

    /* Disable tracking mode */
    error = GroupAnalogTrackingModeDisable (SocketID, pGroup);
    if (0 != error)
    {
        DisplayErrorAndClose(error, SocketID, "GroupAnalogTrackingModeDisable");
        return;
    }

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



1.3.9 Backlash compensation

Configuration:

Group type	*Number*	*Group name*	*Positioner name*
Single axis	1	SINGLE_AXIS	SINGLE_AXIS.MY_STAGE

Description: This example opens a TCP connection and kills the single axis 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 = "SINGLE_AXIS";
pPositioner = "SINGLE_AXIS.MY_STAGE";
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;
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



```
/* 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");
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



1.3.10 Timer event and global variables

Configuration:

Group type	*Number*	*Group name*	*Positioner name*
Single axis	1	SINGLE_AXIS	SINGLE_AXIS.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 = "SINGLE_AXIS.MY_STAGE";
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);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



```
   /* 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
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



```
# 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"]
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



1.3.11 Pulse generation on trajectory (with gathering)

Configuration:

Group type	*Number*	*Group name*	*Positioner name*
Multiple axes	1	M	M.POSITIONER

Description: This example will show you how to configure the XPS to generate pulses on trajectory, and gather data when a pulse occurs. First, the group is killed, initialized and homed. The gathering is then configured to trigger on every pulse (use of event Always together with TrajectoryPulse). The trajectory is then executed. When finished, the gathering is saved and the event removed. 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.

Example trajectory : (defines 10 trajectory points)

```
1,      0,      .2
1,      .1,     .2
1,      .1,     .2
1,      .1,     .2
1,      .1,     .2
1,      .1,     .2
1,      .1,     .2
1,      .1,     .2
1,      .1,     .2
1,      .1,     0
```

C++ code:

```
#define pGroup "M"                                      // Group name
#define pPositioner "M.POSITIONER"                      // Positioner name
#define pTrajectory "trajectory.pvt"                    // Trajectory file name

#define pEventAlways "Always"                           // Event name for Always
#define pEventPulse "M.PVT.TrajectoryPulse"             // Event name for Trajectory pulses
#define pActionGatheringOneData "GatheringOneData"      // Action name for GatheringOneData
#define pZero "0"                                       // Null parameter

char    pIPAddress[15] = "192.168.33.236";
int     nPort = 5001;
double  dTimeOut = 60;
int     SocketID = -1;
int    eventID;                                         // Event ID
int     error = 0;
char    pGatheredData[32] = "M.POSITIONER.CurrentPosition";
char    pTrigger[128];
sprintf(pTrigger, "%s;%s", pEventAlways,pEventPulse);

/* TCP / IP connection */
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");

/* GroupKill, GroupInitialize and GroupHomeSearch  */
printf ("Initializing group %s\n", pGroup);
error = GroupKill (SocketID, pGroup);
if (0 != error)
{
      DisplayErrorAndClose(error, SocketID, "GroupKill");
      return;
}
error = GroupInitialize (SocketID, pGroup);
if (0 != error)
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



```
   {
         DisplayErrorAndClose(error, SocketID, "GroupInitialize");
         return;
   }
   error = GroupHomeSearch (SocketID, pGroup);
   if (0 != error)
   {
         DisplayErrorAndClose(error, SocketID, "GroupHomeSearch");
         return;
   }

   /* Configure gathering & events */
   printf ("Configure gathering & events\n");
   error = GatheringConfigurationSet (SocketID, 1, pGatheredData);
   if (0 != error)
   {
         DisplayErrorAndClose(error, SocketID, "GatheringConfigurationSet");
         return;
   }
   error = EventExtendedConfigurationTriggerSet (SocketID,2,pTrigger,pZero,pZero,pZero,pZero);
   if (0 != error)
   {
         DisplayErrorAndClose(error, SocketID, "EventExtendedConfigurationTriggerSet");
         return;
   }
   error=EventExtendedConfigurationActionSet(SocketID,1,pActionGatheringOneData,pZero,pZero,pZero,pZero);
   if (0 != error)
   {
         DisplayErrorAndClose(error, SocketID, "EventExtendedConfigurationActionSet");
         return;
   }
   error = EventExtendedStart (SocketID, &eventID);
   if (0 != error)
   {
         DisplayErrorAndClose(error, SocketID, "EventExtendedStart");
         return;
   }

   /* Verify PVT  */
   printf ("Verify PVT : %s\n", pTrajectory);
   error = MultipleAxesPVTVerification (SocketID, pGroup, pTrajectory);
   if (0 != error)
   {
         DisplayErrorAndClose(error, SocketID, "MultipleAxesPVTVerification");
         return;
   }
   double minPos, maxPos, maxVel, maxAcc;
   char trajFile[124];
   error=MultipleAxesPVTVerificationResultGet (SocketID,pPositioner,trajFile,&minPos,&maxPos,&maxVel,
&maxAcc);
   if (0 != error)
   {
         DisplayErrorAndClose(error, SocketID, "MultipleAxesPVTVerificationResultGet");
         return;
   }
   printf("Positioner %s :\n", pPositioner);
   printf("\t-> trajectory file = %s\n", trajFile);
   printf("\t-> minimum position = %d\n", minPos);
   printf("\t-> maximum position = %d\n", maxPos);
   printf("\t-> maximum velocity = %d\n", maxVel);
   printf("\t-> maximum acceleration = %d\n", maxAcc);

   /* Configure pulses  */
   printf ("Configure pulses\n");
   error = MultipleAxesPVTPulseOutputSet (SocketID, pGroup, 1, 10, 1);
   if (0 != error)
   {
         DisplayErrorAndClose(error, SocketID, "MultipleAxesPVTPulseOutputSet");
         return;
   }

   /* Execute trajectory  */
   printf ("Execute trajectory\n");
   error = MultipleAxesPVTExecution (SocketID, pGroup, pTrajectory, 1);
   if (0 != error)
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



```
{
      DisplayErrorAndClose(error, SocketID, "MultipleAxesPVTExecution");
      return;
}

/* Stop and save gathering  */
printf ("Stop and save gathering\n");
error = GatheringStopAndSave (SocketID);
if (0 != error)
{
      DisplayErrorAndClose(error, SocketID, "GatheringStopAndSave");
      return;
}

/* Remove event  */
error = EventExtendedRemove (SocketID, eventID);
if (0 != error)
{
      DisplayErrorAndClose(error, SocketID, "EventExtendedRemove");
      return;
}

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2 Visual Basic 6 Drivers

2.1 Visual Basic 6 with XPS DLL

To write a program for the XPS with Visual Basic 6, you must add the file “XPS_C8_drivers.bas” to your Visual Basic project. The file “XPS_C8_drivers.bas” contains function declarations from the XPS-C8 drivers. It uses the XPS DLL. Therefor, the needed files are :

“XPS_C8_drivers.bas” : Visual basic interface file to declare function prototypes from XPS-C8 driver
“XPS_C8_drivers.lib” : XPS-C8 driver library
“XPS_C8_drivers.dll” : XPS-C8 dynamic link library

They are located in the ../Admin/Public/Drivers/DLL directory of the XPS controller.

The “**XPS_C8_drivers.bas**” file (Visual basic) is built in relation to “XPS_C8_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 "XPS_C8_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 “**XPS_C8_drivers.dll**” must be in the WINDOWS directory or in the current directory of the executed program. Please refer to the XPS 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)”).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



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
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



3 Matlab Drivers

3.1 XPS with Matlab

At first, the XPS 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 XPS library for Matlab

First, you have to load the library into Matlab memory. This is done using the following function : “`xps_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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

3.3 Example of a Matlab program

```
% Load the library
xps_load_drivers ;

% Set connection parameters
IP = '192.168.33.234' ;
Port = 5001 ;
TimeOut = 60.0 ;

% Connect to XPS
socketID = TCP_ConnectToServer (IP, Port, TimeOut) ;

% Check connection
if (socketID < 0)
    disp 'Connection to XPS failed, check IP & Port' ;
    return ;
end

% Define the positioner
group = 'GROUP3' ;
positioner = 'GROUP3.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) ;
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

4 Python Drivers

4.1 XPS with Python

The Python interface to the XPS comes in a file '*XPS_C8_drivers.py'* that describes a class *XPS*, with all the XPS functions declared in it. You will need to have it in the same directory as your Python program. It is located on the XPS 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 XPS_C8_drivers
```

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

4.2 Example of Python program

```
# --------- Python program : XPS controller demonstration -------- #
import XPS_C8_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 = XPS_C8_drivers.XPS()

# Connect to the XPS
socketId = myxps.TCP_ConnectToServer('192.168.33.235', 5001, 20)

# Check connection passed
if (socketId == -1):
     print 'Connection to XPS failed, check IP & Port'
     sys.exit ()

# Add here your personal codes, below for example :
# Define the positioner
group = 'XY'
positioner = group + '.X'

# 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
```

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
	[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 ----------#
```

Newport Corporation
Worldwide Headquarters

North America & Asia:
Newport Corporation
1791 Deere Avenue
Irvine, CA 92606
USA

Tel: (949)-863-3144 or
(800)-222-6440
Fax: (949)-253-1680

Email: sales@newport.com
tech@newport.com

Europe:
Newport / Micro-Contrôle S.A.
11 Rue du Bois Sauvage
F-91055 Evry Cedex
FRANCE

Tel: 33-(0)1-60-91-68-68
Fax: 33-(0)1-60-91-68-69

Email: france@newport-fr.com
tech_europe@newport.com



Newport

Visit Newport Online at: **www.newport.com**

ISO 9001
FM 27207

Newport Corporation, Irvine, California, has been certified compliant with ISO 9001 by the British Standards Institution.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP