CBo, 2011-03-14
A000T0029, valid for C-863.11/C-663.11

C-863.11 and C-663.11 DC Motor Controllers – Adapting Your Custom C-x63.10 Software for Use with the C-x63.11

This document is aimed at customers who want to adapt their customized C-x63.10 controller software for use with the C-x63.11 controller. It describes the main differences between the C-x63.10 and the C-x63.11 with regard to the firmware and DLL functions.

In this document, the following names are used:

Controllers

C-x63.10 stands for C-863.10 and C-663.10.

C-x63.11 stands for C-863.11 and C-663.11.

DLLs

Mercury GCS DLL: Native firmware of C-x63.10 with GCS 1 translation in DLL

PI GCS 2 DLL: GCS 2 firmware of C-x63.11

Whereas the firmware of the C-x63.10 is designed in such a way that GCS commands are translated by a Windows DLL into native Mercury commands, the C-x63.11 can handle the GCS commands directly.

To adapt your custom software for use with the C-x63.11, you will have to change your programming code so that it is compatible with the PI GCS 2 DLL. See below for more information on the differences, new features and required changes.

Basic Differences

- The axis names are no longer limited to single characters but can be a name containing up to 8 characters ([A..Z][0..9]-_).
- A daisy-chain network is no longer regarded as a virtual multi-axis controller. As a result, a separate connection must be established for each controller in the network. When a connection is established, the network scan is no longer obligatory but can be skipped.
- Multi-axis commands must be split for the individual controllers. This does not affect runtime performance since in the “older” DLL the commands were split internally and sent one after the other to the individual controllers. So the timing does not change. It can be even faster since the new firmware allows higher baud rates (up to 115200).
- The C-x63.11 controller can store parameters in the non-volatile memory. CST only needs to be called when a new stage type is connected.
- Since the C-x63.11 stores much more information than the C-x63.10, stages connected to the C-x63.11 must no longer be referenced with every start-up of the host software but only when their state on the controller changes.

- The list of GCS commands that can be stored in macros is no longer limited. Absolute moves are now possible within macros. In addition, new features such as the use of variables and relative conditional jumps have been introduced.
- Any of the stored macros can be used as start-up macro. Changing the start-up macro does no longer require copying the content of the “zero” macro.
- Different DLL: The DLL header is now “PI_GCS2_DLL.h” and function names start with “PI_”.

Functions That Need to Be Replaced

- REF -> FRF
- MNL -> FNL
- MPL -> FPL
- REF? -> TRS?
- IsReferenceOK -> FRF?
- BRA? -> SPA? <axis> 0x1A (has brake)
  BRA? will now report the state of the brake.
- IsReferencing -> IsControllerReady

Functions That Are No Longer Available

- Mercury_DFF/Mercury_qDFF
  You can define the axis unit using controller parameters.
- Mercury_GetInputChannelNames/Mercury_GetOutputChannelNames
  The channels are numbered. The number of available input/output channels can be queried with qTIO (digital input/output channels) and qTAC (analog input channels).
- Mercury_GetRefResult – call qFRF
- Mercury_ReceiveNonGCSString/Mercury_SendNonGCSString
  Native commands are no longer supported. PI_GcsCommandset/PI_GcsGetAnswer and PI_GcsGetAnswerSize provide direct access to the C-x63.11 controller.
- Mercury_IsRecordingMacro – does no longer exist
- Mercury_qTNJ – does no longer exist
  The C-863.11 supports two types of joysticks devices. Both joystick devices are connected to the "Joystick" connector but with different voltages. The C-663.11 supports one joystick device.
- Mercury_DFH/Mercury_qDFH
  You can define the zero position using controller parameters.

New Features

- Data recorder
- Daisy chain compatible with C-867, E-861, …
- Same DLL as for C-867, E-517, E-712, E-725, E-753, …
  The code will be much more portable.
- The C-x63.11 controller can store parameter values in the non-volatile memory. Values stored in non-volatile memory are defaults settings, so that the system can be used in the desired way immediately.
- USB connection does no longer require a virtual COM port, so the controller can be connected by means of its serial number. It is found on any PC/USB port without virtual COM port. (The virtual COM port is still installed, so third-party software supporting only RS-232 can still be used.)
- Variables
- Relative conditional jumps in macros
- Arguments for macros

PI_GCS2_DLL Functions not Applicable to C-863/663

- PI_ConnectNIgpib
- PI_ConnectTCPIP
- PI_ConnectTCPIPByDescription
- PI_AOS
- PI_qAOS
- PI_APG
- PI_qAPG
- PI_ATC
- PI_qATC
- PI_qATS
- PI_ATZ
- PI_qATZ
- PI_AVG
- PI_qAVG
- PI_BDR
- PI_qBDR
- PI_CCL

- PI_qCCL
- PI_CCT
- PI_qCCT
- PI_DBR
- PI_qDBR
- PI_DCO
- PI_qDCO
- PI_DDL
- PI_qDDL
- PI_qDDL_SYNC
- PI_DFH
- PI_qDFH
- PI_DPO
- PI_DTC
- PI_qDTL
- PI_EnableTCPIPScan
- PI_EnumerateTCPIPDevices
- PI_qGWD
- PI_qGWD_SYNC
- PI_qHAR
- PI_HasPosChanged
- PI_qHPV
- PI_IFC
- PI_qIFC
- PI_IFS
- PI_qIFS
- PI_IMP
- PI_IMP_PulseWidth
- PI_qIMP
- PI_INI
- PI_IsGeneratorRunning
- PI_NLM
- PI_qNLM

- PI_OAC
- PI_qOAC
- PI_OAD
- PI_qOAD
- PI_OCD
- PI_qOCD
- PI_ODC
- PI_qODC
- PI_OMA
- PI_qOMA
- PI_OMR
- PI_ONL
- PI_qONL
- PI_OSM
- PI_qOSM
- PI_OSMf
- PI_qOSMf
- PI_qOSN
- PI_qOVF
- PI_OVL
- PI_qOVL
- PI_PGS
- PI_PLM
- PI_qPLM
- PI_qPUN
- PI_REP
- PI_RNP
- PI_RTO
- PI_qRTO
- PI_SCN
- PI_qSCN
- PI_SPI
- PI_qSPI

- PI_SSA
- PI_qSSA
- PI_SSL
- PI_qSSL
- PI_qSSN
- PI_qSTE
- PI_SVA
- PI_qSVA
- PI_SVR
- PI_qTAD
- PI_qTLT
- PI_qTNS
- PI_qTPC
- PI_qTSC
- PI_qTSP (Exception: PI_qTSP is available for the C-663)
- PI_TWC
- PI_qTWG
- PI_TWS
- PI_qTWS
- PI_VCO
- PI_qVCO
- PI_VLS
- PI_qVLS
- PI_VMA
- PI_qVMA
- PI_VMI
- PI_qVMI
- PI_VOL
- PI_qVOL
- PI_WAV_LIN
- PI_WAV_PNT
- PI_qWAV
- PI_WAV_RAMP

- PI_WAV_SIN_P
- PI_WCL
- PI_WGC
- PI_qWGC
- PI_WGO
- PI_qWGO
- PI_WGR
- PI_qWMS
- PI_WOS
- PI_qWOS
- PI_WSL
- PI_qWSL
- PI_WTR
- PI_qWTR

Mercury_GCS_sample_single_axis.cpp -> PI_GCS2_Sample_single_axis.cpp

This sample demonstrates how to connect one stage to one C-863 or C-663 controller. The stage is initialized and moved to the reference position if necessary. After this the stage is moved to 10 random target positions.

The following steps are needed:

- Rename functions “Mercury_” to “PI_”, include PI_GCS2_DLL.h and link to PI_GCS2_DLL.
- Change axis name from “A” to “1”.
- Connect() function: Now baud rates up to 115200 can be used. See the user manual for available baud rates and the DIP switch settings. Connections will be established faster since the scan for other devices is not triggered with the PI_ConnectRS232() and PI_ConnectUSB() functions.
- InitStage() function:
    - INI is no longer needed.
    - The C-x63.11 can store the stage parameters in the non-volatile memory. CST is only needed if the stage type changes. If you use only preconfigured controller, CST is not needed at all. To store the parameters in the non-volatile memory use the WPA command. In the sample code, you will notice that the InitStage() function often only checks the state and skips CST().

- Ensure that the stage referred to in the code corresponds to the physically connected stage.
  Example:
  ```
  if (!InitStage("1", "M-505.4PD"))
              return -1;
  ```
  Note that DC motor stages must be connected with the C-863, and stepper motor stages must be connected with the C-663. Otherwise, irreparable damage can result.

- ReferenceIfNeeded() function:
  - IsReferenceOK() must be replaced by qFRF()
  - qREF() becomes qTRS()
  - REF() becomes FRF()
  - MNL() becomes FNL() (if MPL() is used, this becomes FPL())
  - IsReferencing() needs to be replaced by IsControllerReady()
    Note the inverse logic!
  - The Mercury_GCS_DLL implements a lot of functionality of the GCS firmware. So a restart of the application using the PI_Mercury_DLL is always a kind of controller reboot. Therefore, things like the referenced state get lost when restarting the software. This is no longer true for the GCS2 firmware: In many cases, referencing is no longer needed if the controller is not rebooted. In the sample code, you will notice that the ReferenceIfNeeded() function often only checks the state and skips referencing.
- GetLimits() function: Only functions are renamed.
- Move() function: Only functions are renamed.

Mercury_GCS_sample_two_axes.cpp -> PI_GCS2_Sample_two_axes.cpp

This sample demonstrates how to connect two stages to a daisy-chain network of two C-863 or C-663 controllers. The stages are initialized and moved to the reference position if necessary. After this the stages are moved to 10 random target positions.

- The following steps are needed: Rename functions “Mercury_” to “PI_”, include PI_GCS2_DLL.h and link to PI_GCS2_DLL.
- Change axis names from “A” to “1” and from “B” to “1”. The two axes are now distinguished by controller IDs.
- Connect() function:
  - Now baud rates up to 115200 can be used. See the user manual for available baud rates and the DIP switch settings.
  - The two controllers need to be connected separately. Therefore, two variables (“ID1”, “ID2”) for the IDs are needed.

- InitStage() function:
  - INI is no longer needed.
  - The C-x63.11 can store the stage parameters in the non-volatile memory. CST is only needed if the stage type changes. If you use only preconfigured controllers, CST is not needed at all. To store the parameters in the non-volatile memory use the WPA command. In the sample code, you will notice that the InitStage() function often only checks the state and skips CST().
  - Ensure that the stage referred to in the code corresponds to the physically connected stage.
    Example:
    ```
    if (!InitStage(ID1, "1", "M-505.4PD"))
                return -1;
    if (!InitStage(ID2, "1", "M-111.1DG"))
                return -1;
    ```
    Note that DC motor stages must be connected with the C-863, and stepper motor stages must be connected with the C-663. Otherwise, irreparable damage can result.
- ReferenceIfNeeded() function:
  - IsReferenceOK() must be replaced by qFRF().
  - qREF() becomes qTRS().
  - REF() becomes FRF().
  - MNL() becomes FNL() (if MPL() is used, this becomes FPL()).
  - IsReferencing() needs to be replaced by IsControllerReady().
    Note the inverse logic!
  - The Mercury_GCS_DLL implements a lot of functionality of the GCS firmware. So a restart of the application using the PI_Mercury_DLL is always a kind of controller reboot. Therefore, things like the referenced state get lost when restarting the software. This is no longer true for the GCS2 firmware: In many cases, referencing is no longer needed if the controller is not rebooted. In the sample code, you will notice that the ReferenceIfNeeded() function often only checks the state and skips referencing.
  - Loop over two controllers implemented
  - The controllers can be started independently, stages can be referenced at the same time. Therefore, depending on the application and setup, a lot of time can be saved here.
- GetLimits() function: Only functions are renamed.
- Move() function:
  - One call to MOV() is needed for each axis. Internally, the Mercury_GCS_DLL queued the commands for the single axes. So this is not slower than before.

Sample Code

Below you will find sample code for use with either the Mercury GCS DLL or the PI GCS 2 DLL (single axis and two axis systems). Compare the samples to find out more about the differences in the code.

Mercury GCS Sample Code (Single Axis)

The below sample code is also contained in the *Mercury_GCS_sample_single_axis.cpp* file which can be found in the *Samples* folder on the Mercury product CD.

```
#include "windows.h"
#include "stdio.h"
#include "PI_Mercury_GCS_DLL.h"

int ID = -1;

void ShowError(const char* szMessage)
{
	int error = Mercury_GetError(ID);
	char szError[1024];
	Mercury_TranslateError(error, szError, 1024);
	printf("%s - error %d, %s\n", szMessage, error, szError);
}

BOOL Connect()
{
	printf("Connecting with RS-232...\n");
	ID = Mercury_ConnectRS232(1, 9600);
	if (ID<0)
	{
		printf("Could not connect to Mercury(network)!\n");
		return FALSE;
	}

	char buffer[255];
	if (!Mercury_qIDN(ID, buffer, 255))
	{
		ShowError("Could not read *IDN?");
		return FALSE;
	}
	printf("Connected to: %s\n", buffer);

	if (!Mercury_qSAI_ALL(ID, buffer, 255))
	{
		ShowError("Could not read SAI? ALL");
		return FALSE;
	}
	printf("Connected axes: %s\n", buffer);

	return TRUE;
}
```

```
BOOL InitStage(const char* szAxis, const char* szStage )
{
	printf("Init axis %s, stage type %s\n", szAxis, szStage);
	if (!Mercury_CST(ID, szAxis, szStage))
	{
		ShowError("CST failed");
		return FALSE;
	}

	if (!Mercury_INI(ID, szAxis))
	{
		ShowError("INI failed");
		return FALSE;
	}

	return TRUE;
}

BOOL ReferenceIfNeeded(const char* szAxis)
{
	BOOL bFlag;
	if (!Mercury_IsReferenceOK(ID, szAxis, &bFlag))
	{
		ShowError("IsReferenceOK failed");
		return FALSE;
	}
	if (bFlag)
	{
		printf("Axis %s already referenced\n", szAxis);
		return TRUE;
	}

	if (!Mercury_qREF(ID, szAxis, &bFlag))
	{
		ShowError("qREF failed");
		return FALSE;
	}
	if (bFlag) // stage has reference switch
	{
		if (!Mercury_REF(ID, szAxis))
		{
			ShowError("REF failed");
			return FALSE;
		}
		printf("Reference stage for axis %s by reference switch ", szAxis);
	}
	else
	{
		if (!Mercury_qLIM(ID, szAxis, &bFlag))
		{
			ShowError("qLIM failed");
			return FALSE;
		}
```

```
            if (bFlag)
            {
                  if (!Mercury_MNL(ID, szAxis))
                  {
                        ShowError("MNL failed");
                        return FALSE;
                  }
                  printf("Reference stage for axis %s by negative limit switch ",
szAxis);
            }
            else
            {
                  printf("Stage for axis %s has no reference nor limit
switch!\n");
                  return FALSE;
            }

      }

      do
      {
            Sleep(500);
            if (!Mercury_IsReferencing(ID, NULL, &bFlag))
            {
                  ShowError("IsReferencing failed");
                  return FALSE;
            }
            printf(".");
      } while (bFlag);
      printf("\n");

      if (!Mercury_IsReferenceOK(ID, szAxis, &bFlag))
      {
            ShowError("IsReferenceOK failed");
            return FALSE;
      }
      if (bFlag)
      {
            printf("Axis %s successfully referenced!\n", szAxis);
            return TRUE;
      }
      else
      {
            printf("Axis %s not referenced!\n", szAxis);
            ShowError("Reference failed");
            return FALSE;
      }
}

BOOL GetLimits(const char* szAxis, double* travelMin, double* travelMax)
{
      if (!Mercury_qTMN(ID, szAxis, travelMin))
      {
            ShowError("qTMN failed");
```

```
		return FALSE;
	}
	if (!Mercury_qTMX(ID, szAxis, travelMax))
	{
		ShowError("qTMX failed");
		return FALSE;
	}

	return TRUE;
}

BOOL Move(const char* szAxis, double target)
{
	printf("Moving axis %s to %f\n", szAxis, target);
	if (!Mercury_MOV(ID, szAxis, &target))
	{
		ShowError("MOV failed");
		return FALSE;
	}
	BOOL bFlag = TRUE;
	double position;
	do
	{
		Sleep(500);
		if (!Mercury_IsMoving(ID, szAxis, &bFlag))
		{
			ShowError("IsMoving failed");
			return FALSE;
		}
		if (!Mercury_qPOS(ID, szAxis, &position))
		{
			ShowError("qPOS failed");
			return FALSE;
		}
		printf("cur.pos.: %s - %f\n", szAxis, position);
	} while (bFlag);
	printf("\n");
	return TRUE;
}

int main(int argc, char* argv[])
{
	printf("Mercury GCS sample - using PI_Mercury_GCS.dll\n\n");
	if (!Connect())
		return -1;

	if (!InitStage("A", "M-505.4PD"))
		return -1;

	if (!ReferenceIfNeeded("A"))
		return -1;

	double travelMin, travelMax;
	if (!GetLimits("A", &travelMin, &travelMax))
```

```
            return -1;

      double range = travelMax - travelMin;

      for (int i= 0; i<10; i++)
      {
            double target = travelMin + (double(rand())/RAND_MAX)*range;
            if (!Move("A", target))
                  return -1;
      }
      return 0;
}
```

PI GCS 2 Sample Code (Single Axis)

The below sample code is also contained in the *PI_GCS2_sample_single_axis.cpp* file which can be found in the *Samples* folder on the Mercury product CD.

```
#include "windows.h"
#include <stdio.h>
#include "PI_GCS2_DLL.h"

int ID = -1;

void ShowError(const char* szMessage)
{
      int error = PI_GetError(ID);
      char szError[1024];
      PI_TranslateError(error, szError, 1024);
      printf("%s - error %d, %s\n", szMessage, error, szError);
}

BOOL Connect()
{
      printf("Connecting with RS-232...\n");
      ID = PI_ConnectRS232(1, 115200);
      if (ID<0)
      {
            printf("Could not connect to Mercury(network)!\n");
            return FALSE;
      }

      char buffer[255];
      if (!PI_qIDN(ID, buffer, 255))
      {
            ShowError("Could not read *IDN?");
            return FALSE;
      }
      printf("Connected to: %s\n", buffer);

      if (!PI_qSAI_ALL(ID, buffer, 255))
      {
```

```
            ShowError("Could not read SAI? ALL");
            return FALSE;
      }
      printf("Connected axes: %s\n", buffer);

      return TRUE;
}

BOOL InitStage(const char* szAxis, const char* szStage )
{
      printf("Init axis %s, stage type %s\n", szAxis, szStage);
      char buffer[255];
      if (!PI_qCST(ID, szAxis, buffer, 255))
      {
            ShowError("qCST failed");
            return FALSE;
      }
      char* pStage = strchr(buffer, '=');
      pStage++;
      if (strnicmp(szStage, pStage, strlen(szStage))==0)
      {
            printf("stage type already defined\n");
            return TRUE;
      }

      if (!PI_CST(ID, szAxis, szStage))
      {
            ShowError("CST failed");
            return FALSE;
      }

      return TRUE;
}

BOOL ReferenceIfNeeded(const char* szAxis)
{
      BOOL bFlag = TRUE;
      if (!PI_SVO(ID, szAxis, &bFlag))
      {
            ShowError("SVO failed");
            return FALSE;
      }

      if (!PI_qFRF(ID, szAxis, &bFlag))
      {
            ShowError("qFRF failed");
            return FALSE;
      }
      if (bFlag)
      {
            printf("Axis %s already referenced\n", szAxis);
            return TRUE;
      }
```

```
	if (!PI_qTRS(ID, szAxis, &bFlag))
	{
		ShowError("qTRS failed");
		return FALSE;
	}
	if (bFlag) // stage has reference switch
	{
		if (!PI_FRF(ID, szAxis))
		{
			ShowError("REF failed");
			return FALSE;
		}
		printf("Reference stage for axis %s by reference switch ", szAxis);
	}
	else
	{
		if (!PI_qLIM(ID, szAxis, &bFlag))
		{
			ShowError("qLIM failed");
			return FALSE;
		}
		if (bFlag)
		{
			if (!PI_FNL(ID, szAxis))
			{
				ShowError("MNL failed");
				return FALSE;
			}
			printf("Reference stage for axis %s by negative limit switch ",
szAxis);
		}
		else
		{
			printf("Stage for axis %s has no reference nor limit
switch!\n");
			return FALSE;
		}

	}

	do
	{
		Sleep(500);
		if (!PI_IsControllerReady(ID, &bFlag))
		{
			ShowError("IsControllerReady failed");
			return FALSE;
		}
		printf(".");
	} while (!bFlag);
	printf("\n");

	if (!PI_qFRF(ID, szAxis, &bFlag))
	{
```

```
			ShowError("qFRF failed");
			return FALSE;
		}
		if (bFlag)
		{
			printf("Axis %s successfully referenced!\n", szAxis);
			return TRUE;
		}
		else
		{
			printf("Axis %s not referenced!\n", szAxis);
			ShowError("Reference failed");
			return FALSE;
		}
}

BOOL GetLimits(const char* szAxis, double* travelMin, double* travelMax)
{
		if (!PI_qTMN(ID, szAxis, travelMin))
		{
			ShowError("qTMN failed");
			return FALSE;
		}
		if (!PI_qTMX(ID, szAxis, travelMax))
		{
			ShowError("qTMX failed");
			return FALSE;
		}

		return TRUE;
}

BOOL Move(const char* szAxis, double target)
{
		printf("Moving axis %s to %f\n", szAxis, target);
		if (!PI_MOV(ID, szAxis, &target))
		{
			ShowError("MOV failed");
			return FALSE;
		}
		BOOL bFlag = TRUE;
		double position;
		do
		{
			Sleep(500);
			if (!PI_IsMoving(ID, szAxis, &bFlag))
			{
				ShowError("IsMoving failed");
				return FALSE;
			}
			if (!PI_qPOS(ID, szAxis, &position))
			{
				ShowError("qPOS failed");
				return FALSE;
```

```
            }
            printf("cur.pos.: %s - %f\n", szAxis, position);
      } while (bFlag);
      printf("\n");
      return TRUE;
}

int main(int argc, char* argv[])
{
      printf("PI GCS2 sample - using PI_GCS2_DLL.dll\n\n");
      if (!Connect())
            return -1;

      if (!InitStage("1", "M-505.4PD"))
            return -1;

      if (!ReferenceIfNeeded("1"))
            return -1;

      double travelMin, travelMax;
      if (!GetLimits("1", &travelMin, &travelMax))
            return -1;

      double range = travelMax - travelMin;

      for (int i= 0; i<10; i++)
      {
            double target = travelMin + (double(rand())/RAND_MAX)*range;
            if (!Move("1", target))
                  return -1;
      }
      return 0;
}
```

Mercury GCS Sample Code (Two Axes)

The below sample code is also contained in the *Mercury_GCS_sample_two_axes.cpp* file which can be found in the *Samples* folder on the Mercury product CD.

```
#include "windows.h"
#include "stdio.h"
#include "PI_Mercury_GCS_DLL.h"

int ID = -1;

void ShowError(const char* szMessage)
{
	int error = Mercury_GetError(ID);
	char szError[1024];
	Mercury_TranslateError(error, szError, 1024);
	printf("%s - error %d, %s\n", szMessage, error, szError);
}

BOOL Connect()
{
	printf("Connecting with RS-232...\n");
	ID = Mercury_ConnectRS232(1, 9600);
	if (ID<0)
	{
		printf("Could not connect to Mercury(network)!\n");
		return FALSE;
	}

	char buffer[255];
	if (!Mercury_qIDN(ID, buffer, 255))
	{
		ShowError("Could not read *IDN?");
		return FALSE;
	}
	printf("Connected to: %s\n", buffer);

	if (!Mercury_qSAI_ALL(ID, buffer, 255))
	{
		ShowError("Could not read SAI? ALL");
		return FALSE;
	}
	printf("Connected axes: %s\n", buffer);

	return TRUE;
}

BOOL InitStage(const char* szAxis, const char* szStage )
{
	printf("Init axis %s, stage type %s\n", szAxis, szStage);
	if (!Mercury_CST(ID, szAxis, szStage))
	{
		ShowError("CST failed");
		return FALSE;
```

```
	}

	if (!Mercury_INI(ID, szAxis))
	{
		ShowError("INI failed");
		return FALSE;
	}

	return TRUE;
}

BOOL ReferenceIfNeeded(const char* szAxis)
{
	BOOL bFlag;
	if (!Mercury_IsReferenceOK(ID, szAxis, &bFlag))
	{
		ShowError("IsReferenceOK failed");
		return FALSE;
	}
	if (bFlag)
	{
		printf("Axis %s already referenced\n", szAxis);
		return TRUE;
	}

	if (!Mercury_qREF(ID, szAxis, &bFlag))
	{
		ShowError("qREF failed");
		return FALSE;
	}
	if (bFlag) // stage has reference switch
	{
		if (!Mercury_REF(ID, szAxis))
		{
			ShowError("REF failed");
			return FALSE;
		}
		printf("Reference stage for axis %s by reference switch ", szAxis);
	}
	else
	{
		if (!Mercury_qLIM(ID, szAxis, &bFlag))
		{
			ShowError("qLIM failed");
			return FALSE;
		}
		if (bFlag)
		{
			if (!Mercury_MNL(ID, szAxis))
			{
				ShowError("MNL failed");
				return FALSE;
			}
```

```
                printf("Reference stage for axis %s by negative limit switch ",
szAxis);
            }
            else
            {
                printf("Stage for axis %s has no reference nor limit
switch!\n");
                return FALSE;
            }

      }

      do
      {
            Sleep(500);
            if (!Mercury_IsReferencing(ID, NULL, &bFlag))
            {
                ShowError("IsReferencing failed");
                return FALSE;
            }
            printf(".");
      } while (bFlag);
      printf("\n");

      if (!Mercury_IsReferenceOK(ID, szAxis, &bFlag))
      {
            ShowError("IsReferenceOK failed");
            return FALSE;
      }
      if (bFlag)
      {
            printf("Axis %s successfully referenced!\n", szAxis);
            return TRUE;
      }
      else
      {
            printf("Axis %s not referenced!\n", szAxis);
            ShowError("Reference failed");
            return FALSE;
      }
}

BOOL GetLimits(const char* szAxes, double* travelMin, double* travelMax)
{
      if (!Mercury_qTMN(ID, szAxes, travelMin))
      {
            ShowError("qTMN failed");
            return FALSE;
      }
      if (!Mercury_qTMX(ID, szAxes, travelMax))
      {
            ShowError("qTMX failed");
            return FALSE;
      }
```

```
	return TRUE;
}

BOOL Move(const char* szAxis1, double target1, const char* szAxis2, double
target2)
{
	printf("Moving axes %s to %f and %s to %f\n", szAxis1, target1, szAxis2,
target2);
	char szAxes[10];
	szAxes[0] = szAxis1[0];
	szAxes[1] = szAxis2[0];
	szAxes[2] = '\0';
	double targets[2] = { target1, target2 };
	if (!Mercury_MOV(ID, szAxes, targets))
	{
		ShowError("MOV failed");
		return FALSE;
	}
	BOOL bFlag = TRUE;
	double position[2];
	do
	{
		Sleep(500);
		if (!Mercury_IsMoving(ID, NULL, &bFlag))
		{
			ShowError("IsMoving failed");
			return FALSE;
		}
		if (!Mercury_qPOS(ID, szAxes, position))
		{
			ShowError("qPOS failed");
			return FALSE;
		}
		printf("cur.pos.: %s - %f ; %s - %f\n", szAxis1, position[0], szAxis2,
position[1]);
	} while (bFlag);
	printf("\n");
	return TRUE;
}

int main(int argc, char* argv[])
{
	printf("Mercury GCS sample - using PI_Mercury_GCS.dll\n\n");
	if (!Connect())
		return -1;

	if (!InitStage("A", "M-505.4PD"))
		return -1;
	if (!InitStage("B", "M-111.1DG"))
		return -1;

	if (!ReferenceIfNeeded("A"))
		return -1;
```

```
	if (!ReferenceIfNeeded("B"))
		return -1;

	double travelMin[2], travelMax[2];
	if (!GetLimits("AB", travelMin, travelMax))
		return -1;

	double range[2];
	range[0] = travelMax[0] - travelMin[0];
	range[1] = travelMax[1] - travelMin[1];

	for (int i= 0; i<10; i++)
	{
		double target[2];
		for (int axis = 0; axis<2; axis++)
		{
			target[axis] = travelMin[axis] +
(double(rand())/RAND_MAX)*range[axis];
		}
		if (!Move("A", target[0], "B", target[1]))
			return -1;
	}

	Mercury_CloseConnection(ID);
	return 0;
}
```

PI GCS 2 Sample Code (Two Axes)

The below sample code is also contained in the *PI_GCS2_sample_two_axes.cpp* file which can be found in the *Samples* folder on the Mercury product CD.

```
#include "windows.h"
#include "stdio.h"
#include "PI_GCS2_DLL.h"

int ID1 = -1;
int ID2 = -1;

void ShowError(int ID, const char* szMessage)
{
	int error = PI_GetError(ID);
	char szError[1024];
	PI_TranslateError(error, szError, 1024);
	printf("%s - error %d, %s\n", szMessage, error, szError);
}

BOOL Connect()
{
	printf("Connecting with RS-232...\n");
	long nrDevices;
	char szDevices[16*128];
```

```
	int daisyChain = PI_OpenRS232DaisyChain(1, 115200, &nrDevices, szDevices,
16*128);
	ID1 = PI_ConnectDaisyChainDevice(daisyChain, 1);
	if (ID1<0)
	{
		printf("Could not connect to C-863 1!\n");
		return FALSE;
	}

	ID2 = PI_ConnectDaisyChainDevice(daisyChain, 2);
	if (ID2<0)
	{
		printf("Could not connect to C-863 2!\n");
		return FALSE;
	}

	char buffer[255];
	if (!PI_qIDN(ID1, buffer, 255))
	{
		ShowError(ID1, "Could not read *IDN?");
		return FALSE;
	}
	printf("Connected to device 1: %s\n", buffer);
	if (!PI_qIDN(ID2, buffer, 255))
	{
		printf("\n");
		ShowError(ID2, "Could not read *IDN?");
		return FALSE;
	}
	printf("and to device 2: %s\n", buffer);

	if (!PI_qSAI_ALL(ID1, buffer, 255))
	{
		ShowError(ID1, "Could not read SAI? ALL");
		return FALSE;
	}
	printf("Connected axes 1: %s\n", buffer);

	if (!PI_qSAI_ALL(ID2, buffer, 255))
	{
		ShowError(ID2, "Could not read SAI? ALL");
		return FALSE;
	}
	printf("Connected axes 2: %s\n", buffer);

	return TRUE;
}

BOOL InitStage(int ID, const char* szAxis, const char* szStage )
{
	printf("Device %d, Init axis %s, stage type %s\n", ID, szAxis, szStage);
	char buffer[255];
	if (!PI_qCST(ID, szAxis, buffer, 255))
	{
```

```
			ShowError(ID, "qCST failed");
			return FALSE;
		}
		char* pStage = strchr(buffer, '=');
		pStage++;
		if (strnicmp(szStage, pStage, strlen(szStage))==0)
		{
			printf("stage type already defined\n");
			return TRUE;
		}

		if (!PI_CST(ID, szAxis, szStage))
		{
			ShowError(ID, "CST failed");
			return FALSE;
		}

		return TRUE;
}

BOOL ReferenceIfNeeded(int devID1, const char* szAxis1, int devID2, const char*
szAxis2)
{
		int ids[2];
		ids[0] = devID1;
		ids[1] = devID2;
		const char* szAxes[2];
		szAxes[0] = szAxis1;
		szAxes[1] = szAxis2;

		BOOL bRefOK[2];
		int axis;
		for (axis = 0; axis<2; axis++)
		{
			BOOL bFlag = TRUE;
			if (!PI_SVO(ids[axis], szAxes[axis], &bFlag))
			{
				ShowError(ids[axis], "SVO failed");
				return FALSE;
			}

			if (!PI_qFRF(ids[axis], szAxes[axis], &bRefOK[axis]))
			{
				ShowError(ids[axis], "qFRF failed");
				return FALSE;
			}
			if (bRefOK[axis])
			{
				printf("device %d, Axis %s already referenced\n", ids[axis],
szAxes[axis]);
			}
		}

		if (bRefOK[0] && bRefOK[1])
```

```
            return TRUE;

      for(axis=0; axis<2; axis++)
      {
            if (bRefOK[axis])
                  continue;

            BOOL bFlag = FALSE;
            if (!PI_qTRS(ids[axis], szAxes[axis], &bFlag))
            {
                  ShowError(ids[axis], "qTRS failed");
                  return FALSE;
            }
            if (bFlag) // stage has reference switch
            {
                  if (!PI_FRF(ids[axis], szAxes[axis]))
                  {
                        ShowError(ids[axis], "REF failed");
                        return FALSE;
                  }
                  printf("device %d, Reference stage for axis %s by reference
switch ", ids[axis], szAxes[axis]);
            }
            else
            {
                  if (!PI_qLIM(ids[axis], szAxes[axis], &bFlag))
                  {
                        ShowError(ids[axis], "qLIM failed");
                        return FALSE;
                  }
                  if (bFlag)
                  {
                        if (!PI_FNL(ids[axis], szAxes[axis]))
                        {
                              ShowError(ids[axis], "MNL failed");
                              return FALSE;
                        }
                        printf("device %d, Reference stage for axis %s by negative
limit switch ", ids[axis], szAxes[axis]);
                  }
                  else
                  {
                        printf("device %d, Stage for axis %s has no reference nor
limit switch!\n", ids[axis], szAxes[axis]);
                        return FALSE;
                  }

            }
      }

      do
      {
            Sleep(500);
            for(axis=0; axis<2; axis++)
```

```
		{
			if (!PI_IsControllerReady(ids[axis], &bRefOK[axis]))
			{
				ShowError(ids[axis], "IsControllerReady failed");
				return FALSE;
			}
		}
		printf(".");
	} while (! (bRefOK[0]&&bRefOK[1]));
	printf("\n");

	for(axis=0; axis<2; axis++)
	{
		if (!PI_qFRF(ids[axis], szAxes[axis], &bRefOK[axis]))
		{
			ShowError(ids[axis], "qFRF failed");
			return FALSE;
		}
		if (bRefOK[axis])
		{
			printf("device %d, Axis %s successfully referenced!\n",
ids[axis], szAxes[axis]);
			return TRUE;
		}
		else
		{
			printf("device %d, Axis %s not referenced!\n", ids[axis],
szAxes[axis]);
			ShowError(ids[axis], "Reference failed");
			return FALSE;
		}
	}
	return TRUE;
}

BOOL GetLimits(const char* szAxis1, const char* szAxis2, double* travelMin,
double* travelMax)
{
	int ids[2];
	ids[0] = ID1;
	ids[1] = ID2;
	const char* szAxes[2];
	szAxes[0] = szAxis1;
	szAxes[1] = szAxis2;
	for (int axis = 0; axis<2; axis++)
	{
		if (!PI_qTMN(ids[axis], szAxes[axis], &travelMin[axis]))
		{
			ShowError(ids[axis], "qTMN failed");
			return FALSE;
		}
		if (!PI_qTMX(ids[axis], szAxes[axis], &travelMax[axis]))
		{
			ShowError(ids[axis], "qTMX failed");
```

```
                  return FALSE;
            }
      }
      return TRUE;
}

BOOL Move(const char* szAxis1, double target1, const char* szAxis2, double
target2)
{
      printf("Moving axes %s to %f and %s to %f\n", szAxis1, target1, szAxis2,
target2);
      int ids[2];
      ids[0] = ID1;
      ids[1] = ID2;
      const char* szAxes[2];
      szAxes[0] = szAxis1;
      szAxes[1] = szAxis2;
      double targets[2] = { target1, target2 };
      int axis;
      for (axis = 0; axis<2; axis++)
      {
            if (!PI_MOV(ids[axis], szAxes[axis], &targets[axis]))
            {
                  ShowError(ids[axis], "MOV failed");
                  return FALSE;
            }
      }
      BOOL bFlag[2] = {TRUE, TRUE};
      double position[2];
      do
      {
            Sleep(500);
            for (axis = 0; axis<2; axis++)
            {
                  if (!PI_IsMoving(ids[axis], NULL, &bFlag[axis]))
                  {
                        ShowError(ids[axis], "IsMoving failed");
                        return FALSE;
                  }
                  if (!PI_qPOS(ids[axis], szAxes[axis], &position[axis]))
                  {
                        ShowError(ids[axis], "qPOS failed");
                        return FALSE;
                  }
            }
            printf("cur.pos.: %s - %f ; %s - %f\n", szAxis1, position[0], szAxis2,
position[1]);
      } while (bFlag[0] || bFlag[1]);
      printf("\n");
      return TRUE;
}

int main(int argc, char* argv[])
{
```

```
	printf("PI GCS2 sample - using PI_GCS2_DLL.dll\n\n");
	if (!Connect())
		return -1;

	if (!InitStage(ID1, "1", "M-505.4PD"))
		return -1;
	if (!InitStage(ID2, "1", "M-111.1DG"))
		return -1;

	if (!ReferenceIfNeeded(ID1, "1", ID2, "1"))
		return -1;

	double travelMin[2], travelMax[2];
	if (!GetLimits("1", "1", travelMin, travelMax))
		return -1;

	double range[2];
	range[0] = travelMax[0] - travelMin[0];
	range[1] = travelMax[1] - travelMin[1];

	for (int i= 0; i<10; i++)
	{
		double target[2];
		for (int axis = 0; axis<2; axis++)
		{
			target[axis] = travelMin[axis] +
(double(rand())/RAND_MAX)*range[axis];
		}
		if (!Move("1", target[0], "1", target[1]))
			break;
	}

	PI_CloseConnection(ID1);
	PI_CloseConnection(ID2);
	return 0;

}
```