Alpao SDK 4

Programmer Guide

This manual describes how to use the Alpao SDK4 function library when creating programs for your software applications.

Table of Contents

1 Introducing Alpao Mirror SDK v4

The Alpao Software Development Kit (SDK) version 4 is the latest software development kit from ALPAO. Supported under a variety of Windows and Linux operating systems, the Alpao SDK allows for rapid development of high-performance optical applications.

It is possible to interface the libraries with various software (C, C++, Matlab, Python and Labview) using dynamic link library and wrappers.

This document focuses on how developers can use the Alpao SDK to allow their application to communicate with Alpao deformable mirrors.

2 Assumed Knowledge

It is assumed that you know at least one of the languages supported by our SDK and a minimum of knowledge about software and computer hardware.

3 List of Terms

Term	Definition
API	Application Programming Interface
ASDK	Alpao SDK
ACS	Alpao Common Software (see namespace chapter 10.2.1)
DE	Drive Electronics
SDK	Software Development Kit

4 Supported hardware interface and restriction

Name	Restrictions
AdLink LPCI-7200S (Low-profile)	❖ Discontinued by AdLink ❖ Need free PCI slot ❖ Cannot handle more than 3GB of memory
AdLink PCI/PCIe-7200	❖ Need free PCI/PCIe slot
AdLink PCI/PCIe-7300	❖ Need free PCI/PCIe slot ❖ Cannot handle more than 3GB of memory
AdLink PCIe-7350	❖ Need free PCIe slot ❖ Need AdLink PCIS-DASK driver upper to 5.10.2
Interface Corp. PEX-292144	❖ Need free PCIe slot
Ethernet	❖ The security rules for the computer can prevent access to the Ethernet interface.
Usb	❖ The USB interface creates a new network interface; the same limitations as Ethernet apply.

Gigabit Ethernet	❖ The security rules for the computer can prevent access to the Ethernet interface.

5 Supported Platforms

Excluding interface specific limitations, the SDK is compatible with the following operating systems and versions:

Operating system	**Supported Versions**
Operating System Support	
Windows	Windows XP 32 bit
	Windows 7, 8, 8.1 and 10 – 32/64 bit
Linux	Linux - 32 and 64 bit
Integrated Development Environment (IDE) and Compiler Support	
Windows	Microsoft Visual Studio 2010
Linux	GCC / G++

6 Configuration file and serial number

All mirror from ALPAO are referenced by a unique serial number, you can find it on the back of the mirror, for example:

S/N : BAL002
P/N : DMH-HSP-15-097-002

Where "***BAL002***" is the serial number.

For each mirror a configuration file is provided. For DEv5 and DEv7, two configuration files (where BXXYYY is the serial number) are given:

- BXXYYY.acfg which is an ASCII files describing the interface.
- BXXYYY which is a binary file.

For DEv8, one file is provided: BXXYYY.json

The configuration files should be in one of these folders:

- The same folder as your application.
- Or by setting the environment variable ACECFG to point to any folder you want (the installer uses this method to point to *C:\programme\Alpao\SDK\config*).
  - See http://support.microsoft.com/kb/310519 for Windows environment variable.
    - Eg: *ACECFG="C:/Programme/Alpao/SDK/config"*
  - Manual of Export / Set command on Linux.
    - Eg: ACECFG="/user/local/AlpaoSDK/config"

Alpao SDK will search the following directories **in this order** to find the configuration files:

- ./, ../, ACECFG, ACEROOT, ./config, ../config, ACECFG/config, ACEROOT/config

Where "./" is the CURRENT DIRECTORY and "../" is the PARENT DIRECTORY.
ACECFG and ACEROOT are environment variables.

Please note that each deformable mirror is calibrated at ALPAO premises. When turned on, the offsets measured by ALPAO are sent to the deformable mirror. The offset currents are specified in the configuration file (and cannot be modified by the user). Like every electronic device, we suggest warming up the deformable mirror before starting to use it.

7 Application programming interface

7.1 Primitives and Classes

You can found all macro and type definition in “asdkType.h”

Primitive	Description
Char	Unsigned integer, 8 bits
UChar	Signed integer, 8 bits
Short	Unsigned integer, 16 bits
UShort	Signed integer, 16 bits
Int	Unsigned integer, 32 bits
UInt	Signed integer, 32 bits
Long	Unsigned integer, 64 bits
ULong	Signed integer, 64 bits
Size_T	Represent the size of any object in bytes
Bool	Boolean type, value can be true / false on C++ or TRUE / FALSE under C.
CString	C style string type
CStrConst	C style const string type
COMPL_STAT	Default enumeration returned by function, value can be SUCCESS or FAILURE
String	(C++ only) Redefinition of STD string

7.2 Macros definition

IS_WIN32	“1” if compiled for Windows, “0” otherwise
IS_64B	“1” if compiled on 64Bit platform, “0” otherwise

8 List of parameters keyword

8.1 DEv7

Keyword	Get	Set	Value	Description
DacReset		X	1	Reset all digital to analog converters of drive electronics.
daqFreq		X	1e3..20e6	Updates the digital to analog conversion rate in Hz (for PEX-292144, AdLink PCIe-7300/7350 and Gigabit Ethernet)
Gain		X	0..8191	Change digital to analog RAW gain or resolution (default is 8191). Apply it once after drive electronics power-on. e.g. Set("Gain", 4095) will divide the stroke by two and increase the resolution by two.
ItfState	X		0/1	Return 1 if PCI interface is busy or 0 otherwise.
LogDump		X	1	Dump the log stack on the standard output.
LastCommand	X		Array	Read last send command.
LogPrintLevel	X	X	0..4	Changes the output level of the logger to the standard output.
mcff		X	Float array	Coefficient in [0, 1] applied to each intermediate command sent by dm.Send to smooth mirror movement. The default value is a ramp from 0 to 1. mcff is an array of NbSteps elements.
NbOfActuator	X		1..*	Get the numbers of actuator for that mirror.
NbSteps		X	1…*	Number of intermediate commands sent by dm.Send to smooth mirror movement. For real-time control application (> 1 kHz), it is recommended to set this value to 1 to improve performances.
ResetOnClose	X	X	Boolean	If true (default), send a reset to the mirror when the DM object is destroyed.
SyncMode		X	0/1	0: Synchronous mode, will return when *send* is done. 1: Asynchronous mode, return immediately after safety checks.
Timeout		X	> 0	For Ethernet, USB and PEX292144 interface only; set the time-out (s); can be set in synchronous mode only (see *SyncMode*).
TriggerIn		X	0/1/2	Set mode of the (optional) input trigger. 0: Disabled, 1: Trig on rising edge or 2: Trig on falling edge.
TriggerMode		X	0/1	Set mode of the (optional) electronics trigger output. 0: Long pulse width or 1: Short pulse width on each command.
UseException	X	X	0/1	Enables or disables the throwing of an exception on error.
VersionInfo	X		> 0	Alpao SDK core version. e.g. 305040164 is SDK v3.05.04.0164 where 0164 is build number.
CfgPath	X		String	Path to the current configuration file.

8.2 DEv8 and DMMs

Keyword	Get	Set	Value	Description
NbOfActuator alias NbOfMode	X		1..*	Get the number of actuators for a DM, the number of modes for a DMM. NbOfMode is an alias for NbOfActuator.
CfgPath	X		String	Path to the current configuration file.
cmd_vector	X	X	Vector of real values	For DM values are in [-1, 1], for DMM, values are in meter PV.
/config/enable_steps	X	X	0..1	When 1, steps are enabled.
/config/steps	X	X	Vector of real values	Vector of coefficients used to compute intermediate commands sent to smooth mirror movement (when dm.Send or /cmd_vector are used). Unused when enable_steps is 0.
/config/step_period_us	X	X	1..1e6	Period in microseconds between each step. Unused when enable_steps is 0. Minimum value is 13.
/error	X		String	Returns last error as a string of character.

9 Mirror command

9.1 DM

Mirror commands are sent in an array of values; the number of element should be equal to the number of actuators.

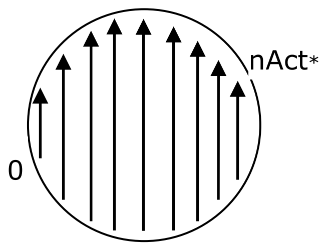
A. Range

All values are standardized and must be between -1 and +1.

B. Drive electronics resolution

Minimum value above zero is approximately 0.0005.

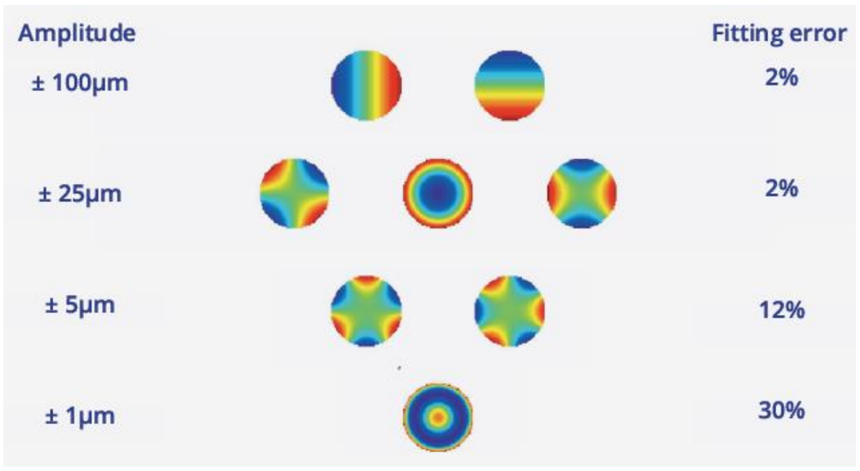
C. Actuator order:



*nAct – Number of mirror actuators

9.2 Modal DM (DMM)

DMM commands are in meter PV for each mode in NOLL order. Warning: a command of 1 mean, 1 meter PV!


Amplitude
Fitting error
± 100µm
2%
± 25µm
2%
± 5µm
12%
± 1µm
30%

10 Language

10.1 Design and Development Guidelines

For all programming languages, there are three common steps:

A. Opening a connection
B. Working with the deformable mirror (Sending commands)
C. Closing the connection

Most methods return a value, which can be checked. Null pointers should always be checked for reference types. For COMPL_STAT, you should test the result for SUCCESS. In most cases, function like *DM::Check()* should be used to test the global status.

We recommend that you become familiar with the other sample applications that are automatically installed into the following directory:

- For Windows: C:\Programme\Alpao\SDK\Samples
- For Linux: ASDK\Samples

The sample applications are installed in these directory locations if your installation process is using the default installation directory.

These sample applications demonstrate most of the functionality available in the Alpao SDK.

10.2 Using C++ compiler

10.2.1 Namespace

To avoid naming collisions, all the SDK names (type, function and class) are wrapped under ACS namespace.

In order to access these names from outside the acs namespace you have to use the scope operator :: . For example, to access the *COMPL_STAT* from outside of acs we can write:

```
acs::COMPL_STAT
```

The keyword *using* is used to introduce a name from a namespace into the current declarative region. For example:

```
{
      using acs::COMPL_STAT;
      COMPL_STAT a;
      COMPL_STAT b;
}
```

You can also access the entire namespace with keyword *using namespace* in a block or in the global scope.

For example:

```
using namespace acs;
int main () {
      COMPL_STAT a;
      DM dm( "SerialNumber" );
}
```

10.2.2 Using one DM per electronics box

In order to use the C++ API with one mirror per electronics, you must include the *asdkDM.h* header. It defines the following class:

```
acs::DM
```

With the following methods:

`DM( CStrConst serialNumber )`		
Default constructor. Use method "DM::Check()" to determine the validity of the object.		
serialNumber	IN	Serial number of the mirror (eg: "BXXYYY").

`COMPL_STAT Send( const Scalar * values )`		
Send value to the mirror, values are normalized between -1 and 1. For DMM, value must be nMode long and contains values in meter PV for nMode Zernike modes (Noll).		
values	IN	Array of values to be sent to the mirror, the number of element should be equal to the number of actuator.
Return	FAILURE in case of failure, SUCCESS otherwise.	

`COMPL_STAT Send( const Scalar * values, UInt nbPattern, UInt nRepeat = 1 )`		
Send patterns to the mirror. Patterns are a set of pre-calculated values sent with greater speed.		
values	IN	Array of values to be sent to the mirror; the number of element should be equal to the number of actuators multiplied by the number of patterns.
nbPattern	IN	Number of patterns to be sent.
nRepeat	IN	Number of time to repeat that pattern (some interface does not allow you to use this feature).
Return	FAILURE in case of failure, SUCCESS otherwise.	

`COMPL_STAT Reset()`	
Set all actuators to the value zero.	
Return	FAILURE in case of failure, SUCCESS otherwise.

`COMPL_STAT Stop()`	
Stops all current transfer (send, pattern ...).	
Return	FAILURE in case of failure, SUCCESS otherwise.

Generic get/set accessors:

`Scalar Get( CStrConst command ) const`		
Get parameter value.		
command	IN	Parameter name, see List of parameters keyword.
Return	If found; return the scalar value of the parameter, you should cast that value to the wanted type. If not found; return 0 and Check() will return false.	

`void Set( CStrConst command, Scalar value )`		
Set parameter value. Use method Check() to determine if the parameter was set correctly.		
command	IN	Parameter name, see List of parameters keyword.
value	IN	Parameter value (numeric).

`void Set( CStrConst command, const Scalar* vector, Int size )`		
Set parameter value as a vector. Use method Check() to determine if the parameter was set correctly.		
command	IN	Parameter name, see List of parameters keyword.
str	IN	Parameter value (null-terminated string).

10.2.3 Using multiple DMs on the same electronics

In order to use the C++ API with multiples mirrors per electronics, you must include the asdkMultiDM.h header. It defines the following class:

```
acs::MultiDM
```

With the following methods:

`MultiDM()`		
Default constructor. Use method "DM::Check()" to determine the validity of the object.		

`COMPL_STAT Add( CStrConst serialNumber )`		
Add one mirror to the list, all mirror should be on the same electronics.		
serialNumber	IN	Serial number of the mirror (eg: "BXXYYY").
Return	FAILURE in case of failure, SUCCESS otherwise.	

`COMPL_STAT Remove( Size_T index )`		
Remove one mirror from the list.		
index	IN	Index of the mirror, from 0 to nDm-1.
Return	FAILURE in case of failure, SUCCESS otherwise.	

`COMPL_STAT Send( const Scalar * values )`		
Send value to the mirror, values are normalized between -1 and 1. For DMM, value must be nMode long and contains values in meter PV for nMode Zernike modes (Noll).		
values	IN	Array of values to be sent to the mirror, the number of element should be equal to the number of actuator.
Return	FAILURE in case of failure, SUCCESS otherwise.	

`COMPL_STAT Reset()`	
Set all actuators to the value zero.	
Return	FAILURE in case of failure, SUCCESS otherwise.

`COMPL_STAT Stop()`	
Stops all current transfer (send, pattern ...).	
Return	FAILURE in case of failure, SUCCESS otherwise.

`COMPL_STAT Send( const Scalar * patterns, UInt nPattern, UInt nRepeat = 1 )`		
Send one pattern to all mirrors. Mirrors should have the same number of actuators. Patterns are a set of pre-calculated values sent with greater speed.		
patterns	IN	Array of values to be sent to the mirror, number of element should be equal to the number of actuator multiplied by the number of patterns.
nPattern	IN	Number of patterns to be sent.
nRepeat	IN	Number of time to repeat that pattern (some interface not allow you to use this feature).
Return	FAILURE in case of failure, SUCCESS otherwise.	

`COMPL_STAT Send( Scalar const* const* patterns, UInt nPatt, UInt nRepeat=1 )`		
Send one pattern per mirror. Patterns are a set of pre-calculated values sent with greater speed.		
patterns	IN	Array of arrays of values to be sent to each mirror, the number of element should be equal to the number of actuator multiplied by the number of patterns, index by the mirror id. `eg: Scalar values[ nDm ][ nAct * nPattern ]`
nPatt	IN	Number of patterns to be sent.
nRepeat	IN	Number of time to repeat that pattern (some interface not allow you to use this feature).
Return	FAILURE in case of failure, SUCCESS otherwise.	

`UInt GetNbOfActuator() const`	
Get the total number of actuators.	
Return	Sum of all mirror actuators.

`UInt GetNbOfDM() const`	
Get the number of mirrors.	
Return	Number of mirrors handled by the object.

Generic get/set accessors:

`Scalar Get( Size_T index, CStrConst command ) const`		
Get parameter value. Use method Check() to determine if the parameter was set correctly.		
index	IN	Index of the mirror, from 0 to nDm-1.
command	IN	Parameter name, see List of parameters keyword.
Return	If found; return the scalar value of the parameter, you should cast that value to the wanted type. If not found; return 0 and Check() will return false.	

`void Set( Size_T index, CStrConst command, Scalar value )`		
Set parameter value. Use method Check() to determine if the parameter was set correctly.		
index	IN	Index of the mirror, from 0 to nDm-1.
command	IN	Parameter name, see List of parameters keyword.
value	IN	Parameter value (numeric).
Return	If found; return the scalar value of the parameter, you should cast that value to the wanted type. If not found; return 0 and Check() will return false.	

`void Set( Size_T index, CStrConst command, CStrConst str)`		
Set parameter value. Use method Check() to determine if the parameter was set correctly.		
index	IN	Index of the mirror, from 0 to nDm-1.
command	IN	Parameter name, see List of parameters keyword.
str	IN	Parameter value (null-terminated string).
Return	If found; return the scalar value of the parameter, you should cast that value to the wanted type. If not found; return 0 and Check() will return false.	

10.2.4 Error handling

Error handling systems is global, so the following methods are the same in class acs::DM and acs::MultiDM:

static Bool Check()	
Global error status for Alpao SDK.	
Return	false if one function fail, true otherwise.

static UInt PrintLastError()	
Pop last message from the stack and print it to the standard output (stdout or stderr).	
Return	Error code (see asdkErrNo.h).

static UInt GetLastError(CString message, Size_T size)		
Get the last error and pop it from the stack. Parameters message can be null and size equal to 0 if you only want to retrieve the error code.		
message	IN/OUT	Buffer to contain the null-terminated string description.
size	IN	Size of the buffer.
Return	Error code (see asdkErrNo.h).	

You can also use the C++ operator "<<" to get the last error message:

```
acs::DM dm( "BXXYYY" );
std::cout << dm << std::endl;
```

10.2.5 Exceptions

By default the Alpao SDK not throws any exception, to be compatible with applications mainly programmed in C.

However, it is possible to enable the exceptions with the command:

```
dm.Set( "UseException", 1 );
```

Or by adding "**UseException 1**" (without quote) to the configuration file BXXYY.acfg.

In that case, the SDK throw an exception when an error is detected and it's no longer necessary to use the Check() method.

```
acs:DM dm("BXXYYY");
if ( !dm ) exit( -1 );
dm.Set( "UseException", 1 );
try
{
    dm.Send( values );
    dm.Stop();
}
catch (std::exception e)
{
  cout << "An exception occurred:" << e.what() << endl;
}
```

10.3 Using C compiler

In order to use the C API, you must include the asdkWrapper.h header. It defines the following functions:

`asdkDM * asdkInit( CStrConst serialName )`		
Initialize connection to the drive electronics.		
serialName	IN	Serial names of the DM.
Return	If success, return the mirror structure; in case of failure return NULL.	

`COMPL_STAT asdkRelease( asdkDM *pDm )`		
Release DM.		
pDm	IN	Pointer to DM structure.
Return	FAILURE in case of failure, SUCCESS otherwise.	

`COMPL_STAT asdkSend( asdkDM *pDm, const Scalar *value )`		
Send value to the mirror, values are normalized between -1 and 1. For DMM, value must be nMode long and contains values in meter PV for nMode Zernike modes (Noll).		
pDm	IN	Pointer to DM structure.
value	IN	Array of values to be sent to the mirror, the number of element should be equal to the number of actuator.
Return	FAILURE in case of failure, SUCCESS otherwise.	

`COMPL_STAT asdkReset( asdkDM *pDm )`		
Reset mirror values.		
pDm	IN	Pointer to DM structure.
Return	FAILURE in case of failure, SUCCESS otherwise.	

`COMPL_STAT asdkSendPattern( asdkDM *pDm, const Scalar *patt, UInt nPatt, UInt nRepeat )`		
Send patterns to the mirror. Patterns are a set of pre-calculated values sent with greater speed. If you use pattern generation with several mirrors, but only single DAQ card, mirror will be queued until the end of preceding pattern execution.		
pDm	IN	Pointer to DM structure.
patt	IN	Array of values to be sent to the mirror; the number of element should be equal to the number of actuators multiplied by the

		number of patterns.
nPatt	IN	Number of patterns to be sent.
nRepeat	IN	Number of time to repeat that pattern (some interface does not allow you to use this feature).
Return	FAILURE in case of failure, SUCCESS otherwise.	

`COMPL_STAT asdkStop( asdkDM *pDm )`		
Stop asynchronous transfer (all mirror on the same interface will be stopped).		
pDm	IN	Pointer to DM structure.
Return	FAILURE in case of failure, SUCCESS otherwise.	

`COMPL_STAT asdkGet( asdkDM *pDm, CStrConst command, Scalar * value )`		
Get value of one parameter.		
pDm	IN	Pointer to DM structure.
command	IN	Parameter name, see List of parameters keyword.
value	OUT	Returned value.
Return	FAILURE in case of failure, SUCCESS otherwise.	

`COMPL_STAT asdkSet( asdkDM *pDm, CStrConst command, Scalar value )`		
Set value of one parameter. N.B. With pex292144 interface, some parameters such as daqFreq are applied when a command is sent. Reading them can lead to undefined value is a command has not been applied.		
pDm	IN	Pointer to DM structure.
command	IN	Parameter name, see List of parameters keyword.
value	IN	New value.
Return	FAILURE in case of failure, SUCCESS otherwise.	

`COMPL_STAT asdkSetVector( asdkDM *pDm, CStrConst command, const Scalar* vector, Int size)`		
Set value of one parameter as a vector.		
pDm	IN	Pointer to DM structure.

command	IN	Parameter name, see List of parameters keyword.
value	IN	New value.
Return	FAILURE in case of failure, SUCCESS otherwise.	

`COMPL_STAT asdkSetString( asdkDM *pDm, CStrConst command, CStrConst cstr )`		
Set value of one parameter (string).		
pDm	IN	Pointer to DM structure.
command	IN	Parameter name, see List of parameters keyword.
cstr	IN	New value.
Return	FAILURE in case of failure, SUCCESS otherwise.	

10.3.1 Error handling

`void asdkPrintLastError()`
Pop last message from the stack and print it to the standard output (stdout or stderr).

`COMPL_STAT asdkGetLastError( UInt *errorNo, CString errMsg, Size_T errSize )`		
Get the last error and pop it from the stack. Parameters message can be null and size equal to 0 if you only want to retrieve the error code.		
errorNo	OUT	Error code.
errMsg	OUT	Buffer to contain the null-terminated string description.
errSize	IN	Size of the buffer.
Return	FAILURE if stack is empty, SUCCESS otherwise.	

10.4 Compatibility with previous SDK

To let you use the same interface as previous version, a compatibility layer has been added. Simply include the header "acedev5.h".

As is a compatibility interface, it will not be described here. Please refer you to the header file for more detail.

11 Compatible application

11.1 Matlab

In order to use the Matlab interface, you must add the folder containing asdkDM.m to your Matlab path. It defines the following class:

```
asdkDM
```

With the following methods:

`obj = asdkDM( serialName )`	
Default class constructor.	
serialName	Serial number of the mirror (eg: "BXXYYY").
Return	Allocated object.

`Send( values )`	
Send values to the mirror.	
values	Vector of nAct values to send.

`Reset()`
Set all actuators to zero.

`SendPattern( values, nRepeat )`	
Send patterns to the mirror. Patterns are a set of pre-calculated values sent with greater speed	
values	Array of values to be sent to the mirror, the number of element should be equal to the number of actuator multiplied by the number of patterns.
nRepeat	Number of time to repeat that pattern (some interface does not allow you to use this feature).

`Stop()`
Stops all current transfer (send, pattern ...).

`value = Get( cmdName )`	
Get parameter value. N.B. With pex292144 interface, some parameters such as daqFreq are applied when a command is sent. Reading them can lead to undefined value is a command has not been applied.	
cmdName	Parameter name, see List of parameters keyword.

Return	Parameter value.

Set(obj, cmdName, value)	
Set parameter value.	
cmdName	Parameter name, see List of parameters keyword.
value	Parameter value (numerical or string).

11.1.1 Error handling

Matlab use Exception by default, so you must use try / catch to handle error:

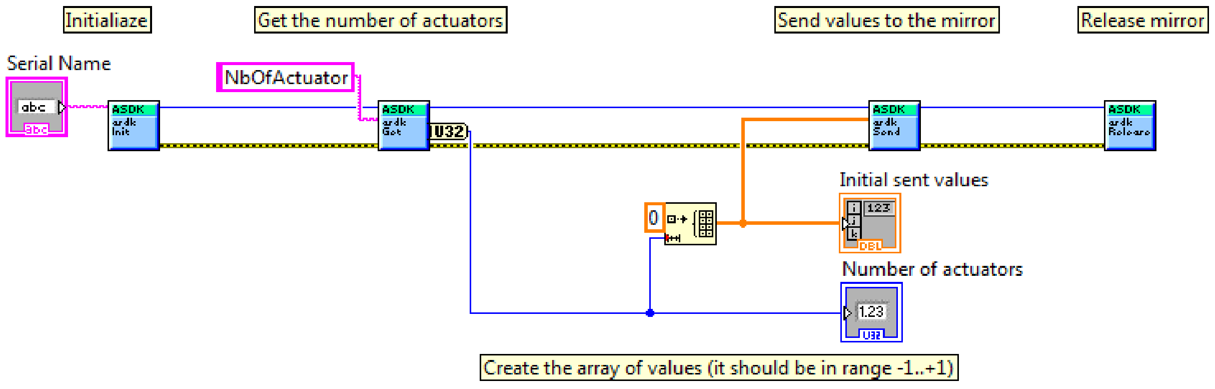
```
try
  dm = asdkDM('nothing');
catch ex
  disp( ['Error with ASDK: ' ex.message ] );
end
```

11.2 Labview

All Labview sub-vi can be found in the "asdk.lib/ASDL.lvlib" file.

You can insert one of the modules by select "insert VI" and select that file.

Simple program is:



12 Known issues

- Adlinks Drivers 5.02 (2012/10/01) can cause some timeout or freeze with PCI-7200 cards.
- Adlinks LPCI-7200S, PCIe-7300 and PCI-7200 cannot handle more than 3GB of memory.
  - Only PCIe-7200 and PCIe-7350 can be used on computer with more memory.

ALPAO S.A.S.
345 RUE LAVOISIER
38330 MONTBONNOT ST MARTIN
FRANCE

☎ **+33 476 890 965**
contact@alpao.com