SPiiPlus .NET Library

Programmer's Guide

May 2025

Document Revision: 4.00e

ACS MOTION CONTROL

TRADEMARKS

Windows and Intellisense are trademarks of Microsoft Corporation.

EtherCAT® is registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany.

Any other companies and product names mentioned herein may be the trademarks of their respective owners.

PATENTS

Israel Patent No. 235022
US Patent Application No. 14/532,023
Europe Patent application No.15187586.1
Japan Patent Application No.: 2015-193179
Chinese Patent Application No.: 201510639732.X
Taiwan(R.O.C.) Patent Application No. 104132118
Korean Patent Application No. 10-2015-0137612

www.acsmotioncontrol.com

support@acsmotioncontrol.com

sales@acsmotioncontrol.com

NOTICE

The information in this document is deemed to be correct at the time of publishing. ACS Motion Control reserves the right to change specifications without notice. ACS Motion Control is not responsible for incidental, consequential, or special damages of any kind in connection with using this document.

Revision History

Date	Revision	Description
Jun 2025	4.10	Updated "ReadStruct" on page 71 and "WriteStruct" on page 72 methods
May 2025	4.00e	- Removed empty examples - Updated ReadStruct and WriteStruct methods - Added missing parameters to FRFOutput
Jan 2025	4.00a	Fixed syntax in SPDataCollectionStart
Dec 2024	4.00	Added support for macOS, see Operating Environment Updated field descriptions in SetServerExtLogin Formatting fixes Updated errors in Error Codes
Aug 2024	3.14.01c	Updated examples in PVT Methods
Jul 2024	3.14.01b	Correct ReadString, other functions from 3.14.01 SetServerExtLogin with blank credentials Add undocumented functions
Jul 2024	3.14.01a	Remove references to undocumented variables
May 2024	3.14.01	New PVT examples Flash write warnings
Mar 2024	3.14a	Update example for StopCollect, NURBS parameter warning
Feb 2024	3.14	Correct parameters for Read/WriteString
Sep 2023	3.13.01.04	Removed async versions of Save and Load Application

Date	Revision	Description
Aug 2023	3.13.01.03	Remove obsolete link
Aug 2023	3.13.01.02	Add previously undocumented functions
Mar 2023	3.13.01.01	Correct ToPoint Example
Feb 2023	3.13.01	New StopCollect example
Nov 2022	3.13	New Release
Jul 2022	3.12.01	Correct AcsplVariableType typo
May 2022	3.12	\r warning on Transaction, new version
Nov 2021	3.11.01	New Version Release
Dec 2020	3.10	Removed enumeration value reserved for internal use only
Sep 2020	3.02	New Release
Jul 2020	3.01	ADK Update
Jun 2020	3.00	New Release
Mar 2019	2.70	> Removed internal use Program Management functions controlling breakpoints

Date	Revision	Description
Jul 2018	2.60	> Replace "X" in programming examples with and axis number, as necessary > Updated all programming examples > Replace extended segmented motion function SegmentArc1 with ExtendedSegment ARC1 SegmentArc2 with ExtendedSegment ARC2
Feb 2018	2.50.10	Replaced "GUI" with "command"
Dec 2017	2.50	Updated for SPiiPlus ADK Suite v2.50
Jun 2017	2.40	Updated for SPiiPlus ADK Suite v2.40
Oct 2016	2.30.01	Removed ACSC_INTR_MESSAGE
Aug 2016	2.30	First release

Conventions

The following conventions are used in the document.

Text Formats

Format	Description
Bold	Names of GUI objects or commands
BOLD + UPPERCASE	ACSPL+ variables and commands
`Monospace + grey background`	Code example
Italics	Names of other documents
Blue	Hyperlink
[ ]	In commands indicates optional item(s)
|	In commands indicates either/or items

Flagged Text

	**Note** - includes additional information or programming tips.
	**Caution** - describes a condition that may result in damage to equipment.
	**Warning** - describes a condition that may result in serious bodily injury or death.

Related Documents

Documents listed in the following table provide additional information related to this document.

The most updated version of the documents can be downloaded by authorized users from www.acsmotioncontrol.com/downloads.

Online versions for all ACS software manuals and SPiiPlus ADK suite Release Notes are available to authorized users at ACS Motion Control Knowledge Center.

Document	Description
SPiiPlus MMI Application Studio User Guide	Explains how to use the SPiiPlus MMI Application Studio and associated monitoring tools.
SPiiPlus Command & Variable Reference Guide	Describes all of the variables and commands available in the ACSPL+ programming language.
SPiiPlus C Library Reference Programmer Guide	C++ and Visual Basic® libraries for host PC applications. This guide is applicable for all the SPiiPlus motion control products
SPiiPlus Utilities User Guide	A guide for using the SPiiPlus User Mode Driver (UMD) for setting up communication with the SPiiPlus motion controller.
SPiiPlus ACSPL+ Programmer's Guide	Provides practical instruction on how to use ACSPL+ to program your motion controller
AN PEG and MARK Operations	Provides detailed description, specification and operation instructions for PEG capabilities.

Table of Contents

List of Tables

1. General Information

1.1 General

The Microsoft .NET has become the most widely used software technology in the world of Windows. The SPiiPlus NET Library provides the easiest way to incorporate SPiiPlus motion control functionality.

1.2 Operating Environment

SPiiPlus ADK Suite Version 3.02 supports the following platforms:

- Microsoft® Windows environments
    - Windows 10 (x64)
    - Windows 11 (x64)
    - Windows Server 2016 R2 (x64)
    - Windows Server 2019 (x64)
    - Windows Server 2022 (x64)
- Linux
    - Ubuntu 18.04
    - Ubuntu 22.04
    - Ubuntu 22.04 (ARM64)
    - Ubuntu 24.04
    - CentOS 7.9
    - AlmaLinux 9.1
- MacOS
    - Sonoma (Apple Silicon ARM64)

1.3 Communication Channels

The SPiiPlus NET Library supports all communication channels provided by SPiiPlus motion controllers:

> Serial (RS-232)

> Ethernet (point-to-point and network)

> PCI Bus

1.4 Controller Simulation

The SPiiPlus NET Library supports communication with a **SPiiPlus Controller Simulator** running on the same PC. The **SPiiPlus Controller Simulator** emulates a controller, enabling program debugging and demonstrations without a connection to a physical controller.

1.5 Supported Programming Languages

The SPiiPlus NET Library supports any programming languages that support the use of .NET components. The SPiiPlus NET Library has been tested with Visual C#.NET applications.

1.6 Installation and Supplied Components

The SPiiPlus NET Library is supplied as a standard .NET assembly (DLL module) and can be references from any .NET application. See Using the SPiiPlus NET Library for details. Samples for.NET are also installed.

1.7 Asynchronous Calls Support

The SPiiPlus NET Library provides an ability of asynchronous operations. Each method that supports asynchronous mode has compliment method with Async postfix. These async methods return ACSC_WAITBLOCK object that can be used to wait for operation completion and receive operation results.

To get a result of a previous asynchronous call GetResult method should be used. It receives the wait block returned by async function and a timeout for wait operation. The Return Value of GetResult method is NET object that can be further cast to string, int or double type, according to asynchronous function called.

ACSC_ WAITBLOCK Waitblock	Wait block returned by Async method.
int timeout	Number of milliseconds to wait for result.
Return Value	When asynchronous execution completes returns result of execution that can be further cast to specific .NET type.

1.8 Standard (SPiiPlus C Library) Features

The SPiiPlus NET Library provides the same standard features as the *SPiiPlus C Library Rererence*, including:

> Unified support for all communication channels (Serial, Ethernet, PCI Bus)

  All SPiiPlus NET Library methods except the **OpenCommxxx** methods are identical for all communication channels. A (host computer) user application doesn't require any other modification to work with different communication channels.

> Controller simulator communication channel

  All SPiiPlus NET Library methods support the SPiiPlus Controller Simulator. The user client application activates the simulator by opening a special communication channel. The user is not required to change his application in order to communicate with the simulator.

> Concurrent support for up to 10 communication channels in one application

  One user client application can open up to 10 communication channels simultaneously. Different communication channels are usually connected to different controllers.

However, two or more communication channels can be connected to the same controller. For *Example*, an application can communicate with a controller through both the controller's Ethernet channel and its serial channels.

> Acknowledgement for each command to the controller

  The library automatically checks the status of each command sent by the user application to the controller. The user application can check the status to confirm that the command was received successfully.

> Communication history

  The SPiiPlus NET Library enables storage in a memory buffer of all messages sent to, and received from, the controller. The application can retrieve data from the buffer and can clear the buffer.

> Separate processing of unsolicited messages

  Most messages sent from the controller to the host are responses to host commands. However, the controller can send unsolicited messages, for Example, as output from a **DISP** command. The library separates the unsolicited messages from the overall message flow and provides a special method for handling unsolicited messages.

> Rich functionality

  The .NET library supports setting and reading parameters, advanced motion control, program management, I/O, safety, and more.

> Debug tools

  The SPiiPlus NET Library provides tools that facilitate debugging of the user application. The simulator and the communication history mentioned above are the primary debugging tools. The user can also open a log file that stores all communications between the application and the controller.

1.9 Specific .NET Library Features

The SPiiPlus NET Library also supports features based on .NET technology:

> Generating events for predefined controller events

  The SPiiPlus NET Library generates events for the user client application if one of the following events occurs:

  > Emergency Stop signal has been generated

  > Motion has ended

  > Motion has been interrupted due to a fault

  > Motor has been disabled due to a fault

  > ACSPL+ program in the controller has completed

  > User-defined event with passed parameters by **INTERRUPTEX** command

  > Other events

> Error handling

The SPiiPlus NET Library raises ACSException with rich error information to a user client application such as error code, Error Description and help context.

1.10 Communication Scenarios

One user application can open up to 10 communication channels simultaneously through the SPiiPlus NET Library. Usually the application opens different communication channels to work with different controller at the same time.

The following communication scenarios are typical for application development.

- Application works with one controller through one communication channel

  Application creates communication channel object, opens communication and then uses any SPiiPlus NET Library methods. Before closing the application closes communication and then destroys the communication channel object.

```
// create communication object
Api channel = new Api();

// open connection (simulator)
channel.OpenCommSimulator();

// get axis position
double position = channel.GetFPosition(Axis.ACSC_AXIS_1);

// close communication
channel.CloseComm();
```

- Application works with several controllers by turns. Only one communication channel is open at the same time

  Application creates communication channel object, opens communication with first controller and then uses any SPiiPlus NET Library methods. When the application needs to connect to the second controller, it closes the previous communication and then opens new a communication. In this case the application uses the same communication channel object for each controller.

  Before closing the application closes communication, then destroys communication channel.

```
// create communication object
Api channel = new Api();

// open connection (simulator)
channel.OpenCommSimulator();

// get axis position
double position = channel.GetFPosition(Axis.ACSC_AXIS_1);

// close communication
channel.CloseComm();
```

```
// open connection (serial)
channel.OpenCommSerial(1, 115200);

// get axis position
position = channel.GetFPosition(Axis.ACSC_AXIS_0);

// close communication
channel.CloseComm();
```

> Application works with several controllers at the same time

In this case the application needs to open several communication channel objects, one for each controller. Then for each communication channel it opens communication with different controllers. Before closing the application closes all communications, then destroys all communication channels.

```
// create communication object
Api channel1 = new Api();
Api channel2 = new Api();
Api channel3 = new Api();

// open connections
channel1.OpenCommSimulator();
channel2.OpenCommSerial(1, 115200);
channel3.OpenCommPCI(Api.ACSC_NONE);

// get axis position
double position1 = channel1.GetFPosition(Axis.ACSC_AXIS_1);
double position2 = channel1.GetFPosition(Axis.ACSC_AXIS_2);
double position3 = channel1.GetFPosition(Axis.ACSC_AXIS_3);

// close communication
channel1.CloseComm();
channel2.CloseComm();
channel3.CloseComm();
```

2. Using the SPiiPlus NET Library

This chapter describes how to use the SPiiPlus NET Library and provides *Example*s of how to call SPiiPlus NET methods from C#.

2.1 Redistribution

SPiiPlus NET Library is installed automatically by the SPiiPlus ADK Suite installation.

2.2 Visual Studio .NET

1. In the “Solution Explorer” window, right click on **Reference** and select “**Add Reference...**”
2. In the **Project** menu, click **References**. The dialog box opens.
3. In the dialog box, click on “Browse” button and select **ACS.SPiiPlusNET.dll**.
4. Declare in your project:

   C#:

   ```
   Using ACS.SPiiPlusNET;
   ```

5. In the method that is called when your form is loaded, include:

   ```
   Api channel = new Api();
   ```

6. Now you can call SPiiPlus NET Library methods. For *Example*,

   ```
   channel.OpenCommSimulator();
   ```

3. Methods

This chapter details the methods available in the SPiiPlus NET Library. For each method there is a:

> Brief description

> Syntax in the form of: **object.method_name(argument, argument, ...)**, where **object** is a placeholder for a valid object name, and each argument is of the form: **type argument_name**.

> Description of the arguments and their function.

> Return Value

> Remarks

> Example - A short C# code example.

The methods are broken down into the following categories:

> Data Collection Methods
> Status Report Methods
> Inputs/Outputs Access Methods
> Safety Control Methods
> Wait for Condition Methods
> Event and Interrupt Handing Methods
> Variables Management Methods
> Service Methods
> Error Diagnostics Methods
> Position Event Generation (PEG) Methods
> Application Save/Load Methods
> Load/Upload Data To/From Controller Methods
> Emergency Stop Methods
> Reboot Methods
> Host-Controller File Operations
> Save to Flash Function

3.1 Communication Methods

SPiiPlus NET Communication methods are as follows:

Table 3-1. Communication Methods

Method	Description
CancelOperation	Cancels all operations.
Command	Sends a command to the controller and analyzes the controller response.
CloseComm	Closes communication (for all kinds of communication).
CloseSimulator	Stops the Simulator in case it is running.
GetConnectionInfo	Retrieves the details of opened communication channel.
GetConnectionList	Retrieves all currently opened on active server connections and its details.
GetEthenetCards	Retrieves all SPiiPlus controller IP addresses within a local domain.
GetPCICards	Retrieves information about the installed SPiiPlus PCI cards.

Method	Description
OpenCommSimulator	Starts up the Simulator and opens communication with the Simulator.
OpenCommEthernetTCP	Opens communication with the controller via Ethernet using the TCP protocol.
OpenCommEthernetUDP	Opens communication with the controller via Ethernet using the UDP protocol.
OpenCommPCI	Opens a communication channel with the controller via PCI Bus.
OpenCommSerial	Opens communication with the controller via a serial port.
SetServerExtLogin	Sets the remote User-Mode Driver (UMD) with a specified IP address, port number, and user login.
TerminateConnection	Terminates specified communication channel (connection) on active server.
Transaction	Executes one transaction with the controller, i.e., it sends a command and receives a controller response.

The Communication methods employ the following structures:

3.1.1 Structures

The Communication methods employ the following structures:

3.1.1.1 ACSC_CONNECTION_DESC

Description

This structure describes the connection information.

Syntax

```
struct ApplicationFileInfo
{
string Application; IntPtr handle;
UInt32 ProcessId;
}
```

Members

Application	Application Name
handle	Channel's handle

ProcessId	Application process ID

3.1.1.2 ACSC_PCI_SLOT

Description

This structure defines a physical location of PCI card.

Syntax

```
struct ACSC_PCI_SLOT
{
UInt32 BusNumber;
UInt32 SlotNumber;
UInt32 Function;
}
```

Members

BusNumber	PCI physical bus number of card
SlotNumber	PCI physical slot number of card
Function	PCI function of card

3.1.1.3 ACSC_CONNECTION_INFO

Description

This structure provides information about specified controller connection for an application. Used in the GetConnectionInfo method.

Syntax

```
struct ACSC_CONNECTION_INFO
{
ACSC_CONNECTION_TYPE Type;
int SerialPort;
int SerialBaudRate;
int PCISlot;
int EthernetProtocol;
string EthernetIP;
int EthernetPort;
}
```

Members

Type	Connection Type
SerialPort	Communication channel of serial communication: 1 corresponds to COM1, 2 – to COM2, etc.

SerialBaudRate	Communication rate of serial communication in bits per second
PCISlot	Number of the PCI slot of the controller card.
EthernetProtocol	Ethernet protocol
EthernetIP	Contains the network address of the controller in symbolic or TCP/IP dotted form.
EthernetPort	Service port.

3.1.2 Enumerations

The Communication methods employ the following enums:

3.1.2.1 ACSC_CONNECTION_TYPE

Description

This enum is used for setting communication type. Used in the **GetConnectionInfo** method.

Values

ACSC_NOT_CONNECTED	Value 0: Not Connected
ACSC_SERIAL	Value 1: Serial Communication
ACSC_PCI	Value 2: PCI Communication
ACSC_ETHERNET	Value 3: Ethernet Communication
ACSC_DIRECT	Value 4: Direct (Simulator) Communication

3.1.3 CancelOperation

Description

This method cancels asynchronous (non-waiting) call.

Syntax

object.CancelOperation(ACSC_WAITBLOCK wait)

Wait	ACSC_WAITBLOCK object returned by invocation of asynchronous call that needed to be canceled

Return Value

None

Remarks

The method waits for the controller response. If the method fails, the Error object is filled with the Error Description.

Example

```
// Shutdown all motors
ACSC_WAITBLOCK wait= Ch.DisableAllAsync();
// Cancel shutdown
Ch.CancelOperation(wait);
```

3.1.4 Command

Description

The method sends a command to the controller and analyzes the controller response.

Syntax

object.Command(string commendName)

Async Syntax

ACSC_WAITBLOCK object.CommandAsync(string commandName)

Arguments

commandName	The command to be sent.

Return Value

None

Remarks

The method sends a string containing one or more ACSPL+ commands to the controller. The method verifies that the controller receives the command but does not return the controller response. The method is intended for commands where the controller response does not include any information except the prompt.

For commands where the controller response includes some information, use Transaction.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Executed controller’s command
 Ch.Command(“enable 0”);
```

3.1.5 CloseComm

Description

The method closes communication via the specified communication channel.

Syntax

object.CloseComm()

Arguments

None

Return Value

None

Remarks

The method closes the communication channel and releases all system resources related to the channel. If the method closes communication with the Simulator, it also terminates the Simulator.

Each **OpenComm*** ** (for *Example*, **OpenCommSerial**) call in the application must be followed at some time with a **CloseComm** call in order to return the resources to the system.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Closes the communication channel
 Ch.CloseComm();
```

3.1.6 CloseSimulator

Description

The method stops the Simulator in case it is running.

Syntax

object.CloseSimulator()

Arguments

None

Return Value

None

Remarks

If the method fails, the Error object is filled with the Error Description.

Example

```
// Closes the Simulator
 Ch.CloseSimulator();
```

3.1.7 GetConnectionInfo

Description

The method is used to retrieve the details of opened communication channel.

Syntax

object.GetConnectionInfo();

Arguments

None

Return Value

ACSC_CONNECTION_INFO.

Example

```
// Get connection info
ACSC_CONNECTION_INFO connInfo = Ch.GetConnectionInfo();
string txtLog = "";
txtLog += "Serial Port: " + connInfo.SerialPort.ToString() + "\n\r";
txtLog += "IP: " + connInfo.EthernetIP.ToString() + "\n\r";
txtLog += "Ethernet Port:" + connInfo.EthernetPort.ToString() + "\n\r";
txtLog += "Ethernet Protocol: " + connInfo.EthernetProtocol.ToString() +
"\n\r";
txtLog += "PCI Slot: " + connInfo.PCISlot.ToString() + "\n\r";
txtLog += "Serial Baud Rate: " + connInfo.SerialBaudRate.ToString() +
"\n\r";
txtLog += "Serial Port: " + connInfo.SerialPort.ToString() + "\n\r";
```

3.1.8 GetConnectionsList

Description

This method retrieves all currently opened connections on active server and their details.

Syntax

object.GetConnectionsList()

Arguments

None

Return Value

ACSC_CONNECTION_DESC[].

The method returns connections Description array. Array length equal to number of active connections found or zero if there are no connections on active server.

Remarks

Each connection is described by its handle, application name and process ID, therefore, all Arguments will have the same size and order, after the method returns.

This method can be used to check if there are some unused connections that remain from applications that did not close the communication channel or were not gracefully closed (terminated or killed).

Each returned connection can be terminated only by the TerminateConnection method.

Using any method of the **SetServer** family makes previously returned connections irrelevant because it changes the active server.

Example

```
// Get active connections list, find application by name and terminate
// its connection
```

```
string name = "neededApplication.exe";
ACSC_CONNECTION_DESC[] connectionList = Ch.GetConnectionsList();
for (int index = 0; index < connectionList.Length; index++)
{
        if (name.Equals(connectionList[index].Application))
         {
                Ch.TerminateConnection(connectionList[index]);
         }
}
```

3.1.9 GetEthernetCards

Description

The method retrieves all SPiiPlus controller IP addresses within a local domain through a standard Ethernet communication protocol.

Syntax

object.GetEthernetCards(IPAddress broadcast)

Arguments

broadcast	IP address for broadcasting. Normally has to be **IPAddress.Broadcast**.

Return Value

IPAddress[].

The method returns array of IP addresses of found SPiiPlus cards or empty array if no cards are detected.

Example

```
IPAddress[] ipAddresses = api.GetEthernetCards(IPAddress.Broadcast);
for (int index = 0; index < ipAddresses.Length; index++)
{
   string address = ipAddresses[index].ToString();
   /*...*/
}
```

3.1.10 GetPCICards

Description

This method retrieves information about the controller cards inserted in the computer PCI Bus.

Syntax

object.GetPCICards()

Arguments

None

Return Value

ACSC_PCI_SLOT[].

The method returns array of found cards.

Remarks

If no controller cards are detected, the method returns empty array, otherwise method fills the array with information about the detected cards.

The **slotNumber** member of ACSC_PCI_SLOT structure can be used in the OpenCommPCI call to open communication with a specific card. Other parameters have no use in the SPiiPlus NET Library.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Example call GetPCICards
ACSC_PCI_SLOT[] cards = Ch.GetPCICards();
// Get info about found cards
for (int index = 0; index < cards.Length; index++)
{
       var busNumber = cards[index].BusNumber;
       var slotNumber = cards[index].SlotNumber;
       var func = cards[index].Function;

/*...*/

}
```

3.1.11 OpenCommSimulator

Description

The method starts up the Simulator and opens communication with the Simulator.

Syntax

object.OpenCommSimulator()
Arguments

None

Return Value

None

Remarks

The Simulator is a part of the SPiiPlus NET Library. When the application calls **OpenCommSimulator**, the library starts up the Simulator as a separate process on the same PC and opens a simulator communication channel for communication with the simulator.

After a channel is open, any SPiiPlus NET method works with the Simulator exactly as with a physical controller.

For the simulator to be to be found by **OpenCommSimulator** a copy of the simulator executable file, **sb4.exe** (located in the SPiiPlus BIN folder) must be in the same folder as the application executable file.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Starts up the Simulator and opens communication with the Simulator
Ch.OpenCommSimulator();
```

3.1.12 OpenCommEthernetTCP

Description

The method opens communication with the controller via Ethernet using the TCP protocol.

Syntax

object.OpenCommEthernetTCP(string address, int port)

Arguments

address	String that contains the network address of the controller in symbolic or TCP/IP dotted form.
port	Service port.

Return Value

None

Remarks

After a channel is open, any SPiiPlus NET method works with the channel irrespective of the physical nature of the channel.

Use TCP protocol if the host computer is connected to the controller through the multi-node Ethernet network.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Open Ethernet with TCP protocol communication with the controller
// IP address: 10.0.0.100
int port = (int)EthernetCommOption.ACSC_SOCKET_STREAM_PORT;
Ch.OpenCommEthernetTCP("10.0.0.100", port);
```

3.1.13 OpenCommEthernetUDP

Description

The method opens communication with the controller via Ethernet using the UDP protocol.

Syntax

object.OpenCommEthernetUDP(string address, int port)

Arguements

address	String that contains the network address of the controller in symbolic or TCP/IP dotted form.
port	Service port.

Return Value

None

Remarks

After a channel is open, any SPiiPlus NET method works with the channel irrespective of the physical nature of the channel.

Use UDP protocol if the host computer has a point-to-point Ethernet connection tothe controller.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Open Ethernet with UDP protocol communication with the controller
// IP address: 10.0.0.100
int port = (int) EthernetCommOption.ACSC_SOCKET_DGRAM_PORT;
Ch.OpenCommEthernetUDP("10.0.0.100", port);
```

3.1.14 OpenCommPCI

Description

The method opens a communication channel with the controller via PCI Bus.

Up to 4-communication channels can be open simultaneously with the same SPiiPlus card through the PCI Bus.

Syntax

object.OpenCommPCI(int SlotNumber)

Arguments

slotNumber	Number of the slot of the controller card. If slotNumber is Api.ACSC_NONE, the method opens communication with the first found controller card.

Return Value

None

Remarks

To open PCI communication the host PC, one or more controller cards must be inserted into the computer PCI Bus.

After a channel is open, any SPiiPlus NET method works with the channel irrespective of the physical nature of the channel.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Open communication with the first found controller card
Ch.OpenCommPCI(Api.ACSC_NONE);
```

3.1.15 OpenCommSerial

Description

The method opens communication with the controller via a serial port.

Syntax

object.OpenCommSerial(int channel, int rate)

Arguments

channel	Communication channel: 1 corresponds to COM1, 2 – to COM2, etc.
rate	Communication rate in bits per second (baud). This parameter must be equal to the controller variable **BAUD** for the successful link with the controller.

Return Value

None

Remarks

After a channel is open, any SPiiPlus NET method works with the channel irrespective of the physical nature of the channel.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Open serial communication through port COM1 with baud rate 115200
Ch.OpenCommSerial(1, 115200);
```

3.1.16 SetServerExtLogin

Description

The method sets the remote User-Mode Driver (UMD) with a specified IP address, port number, and user login credentials.

Syntax

object.SetServerExtLogin(string iP,int port, string username, string password, string domain)

Arguments

iP	IP address of the remote UMD. This address should be the same address displayed in the remote UMD dialog.
port	Remote service port
username	User name on the remote host. Set to "" if credentials are not used.
password	Password on the remote host. Set to "" if credentials are not used.
domain	Domain name on the remote host. Set to "" if credentials are not used.

Return Value

None

Remarks

The method sets the remote UMD with the specified IP address, port number, and user login information. User login information is required if the remote UMD does not run in the current domain, or the remote computer was not previously logged in to the current domain.

Example

```
string ip = "10.0.0.100";
string userName = "USER";
string password = "1234";
string domain = "ACS";
Ch.SetServerExtLogin(ip, (int)GeneralDefinition.ACSC_DEFAULT_REMOTE_PORT,
userName, password, domain);
EthernetCommOption port = EthernetCommOption.ACSC_SOCKET_STREAM_PORT;
```

3.1.17 TerminateConnection

Description

This method terminates specified communication channel (connection) on active server.

Syntax

object.TerminateConnection(ACSC_CONNECTION_DESC connection)

Arguments

connection	Connection description structure obtained through call to **GetConnectionsList** method.

Return Value

None

Remarks

This method can be used to terminate connections that remain from applications that did not close the communication channel or were not gracefully closed (terminated or killed).

Using any method of the **SetServer** family makes previously returned connections list irrelevant because it changes active server, and obligates new call of **GetConnectionsList**.

Example

```
// Get active connections list , find application by name and terminate
//its connection
string name = "neededApplication.exe";
```

3.1.18 Transaction

Description

The method executes one transaction with the controller, i.e., it sends a command and receives a controller response.

Syntax

object.Transaction(string command)

> Any ASCII command being sent to the controller must end with the '\r' (13) character, otherwise it will not be recognized as valid.

Async Syntax

ACSC_WAITBLOCK object.TransactionAsync(string command)

Arguments

command	Command to be sent.

Return Value

String

Remarks

The full operation of transaction includes the following steps:

1. Send **Command** to the controller.
2. Wait until the controller response is received or the timeout occurs.
3. Return a controller response.
4. If the method fails, the Error object is filled with the error Description.

Example

```
// Get controller serial number
String cmd = "?SN";
// The method executes one transaction
String reply = Ch.Transaction(cmd);
```

3.2 EtherCAT® Methods

The EtherCAT methods are:

Table 3-2. EtherCAT Methods

Method	Description
GetEtherCATState	Retrieves the EtherCAT state.
GetEtherCATError	Retrieves the last EtherCAT error code.
MapEtherCATInput	Maps network input variables.
MapEtherCATOutput	Maps network output variables.
UnmapEtherCATInputsOutputs	Cancels mapping of input or output variables.
GetETherCATSlaveIndex	Retrieves index of specified EtherCAT slave.
GetEtherCATSlaveOffsetV2	Retrieves offset of specified EtherCAT slave.
GetETHERCATSlaveVendorID	Retrieves Vendor ID of specified EtherCAT slave.
GetEtherCATSlaveProductID	Retrieves Product ID of specified EtherCAT slave.
GetEtherCATSlaveRevision	Retrieves revision of specified EtherCAT slave.
GetEtherCATSlaveType	Retrieves type of specified EtherCAT slave.
GetEtherCATSlaveState	Retrieves EtherCAT state of specified EtherCAT slave.

3.2.1 GetEtherCATState

Description

The method is used to retrieve the EtherCAT state.

Syntax

object.GetEtherCATState()

Async Syntax

ACSC_WAITBLOCK object.GetEtherCATStateAsync()

Arguments

None

Return Value

EtherCatFlags.

Remarks

The EtherCAT State contained in the variable designated by the State argument is defined by the following bits:

Table 3-3. EtherCAT States

Bit	Name	Description
0	#SCAN	The scan process was performed successfully.
1	#CONFIG	There is no deviation between XML and actual setup.
2	#INITOK	All bus devices are successfully set to INIT state.
3	#CONNECTED	Indicates valid Ethernet cable connection to the master.
4	#INSYNC	Indicates synchronization between master and the bus.
5	#OP	The EtherCAT bus is operational.
6	#DCSYNC	Distributed clocks are synchronized.
7	#RINGMODE	Ring Topology mode status
8	#RINGCOMM	Ring Communication active status
9	#EXTCONN	External clock is connected
10	#DCXSYNC	External clock/slaves are synchronized

All bits (except **#RINGMODE** and **#RINGCOMM** in some cases) should be ON for proper bus functioning. When monitoring the bus state, checking bit #OP is enough. Any bus error will reset the #OP bit.

Example

```
// class="Reference">Example call GetEtherCATState
 EtherCatFlags state = api.GetEtherCatState();
```

3.2.2 GetEtherCATError

Description

The method is used to retrieve the last EtherCAT error code.

Syntax

object.GetEtherCATError()

Async Syntax

ACSC_WAITBLOCK object.GetEtherCATErrorAsync()

Arguments

None

Return Value

Int32.

Remarks

The EtherCAT error codes are:

Table 3-4. EtherCAT Error Codes

Error	Description
6000	General EtherCAT Error
6001	EtherCAT cable not connected
6002	EtherCAT master is in incorrect state
6003	Not all EtherCAT frames can be processed
6004	EtherCAT Slave Error
6005	EtherCAT initialization failure
6007	EtherCAT work count error
6008	Not all EtherCAT slaves are in the OP State
6009	EtherCAT protocol timeout
6010	Slave initialization failed
6011	Bus configuration mismatch
6012	CoE Emergency
6013	EtherCAT Slave won't enter INIT state
6014	EtherCAT ring topology requires network reconfiguration
6015	One or more EtherCAT cables are not connected
6018	EtherCAT Master won't enter PREOP state
6019	EtherCAT Master won't enter SAFEOP state
6020	EtherCAT Master won't enter OP state

Example

```
// Example call GetEtherCATError
int error = Ch.GetEtherCATError();
```

3.2.3 MapEtherCATInput

Description

The method is used for raw mapping of network input variables of any size. Once the method is called successfully, the firmware copies the value of the network input variable at the corresponding EtherCAT offset into the ACSPL+ variable every controller cycle.

> The method call is legal only when the EtherCAT State is OP.

Syntax

object.MapEtherCATInput(MotionFlags flags, int offset, string variableName)

Async Syntax

ACSC_WAITBLOCK object.MapEtherCATInputAsync(MotionFlags flags, int offset, string variableName)

Arguments

flags	Bit-mapped parameter. Currently should be ACSC_NONE.
offset	Internal EtherCAT offset of network input variable. Should be taken from the response to the #ETHERCAT command in the SPiiPlus MMI Application Studio **Communication Terminal** or can be seen via **EtherCAT Configurator**.
variableName	Valid name of ACSPL+ variable, global or standard

Return Value

None

Example

In the following Example, network variable of EtherCAT node 0 at offset 43 is being mapped to the global variable: **IO**.

```
// Example synchronous call MapEtherCATInput
double DBuffer = api.GetDBufferIndex();
api.AppendBuffer((ProgramBuffer)DBuffer,"global int IO");
api.CompileBuffer((ProgramBuffer)DBuffer);
api.MapEtherCATInput(EtherCATFlags.ACSC_NONE, 104, "IO");
```

3.2.4 MapEtherCATOutput

Description

The method is used for raw mapping of network output variables of any size. Once the method is called successfully, the firmware copies the value of specified ACSPL+ variable into the network output variable at the corresponding EtherCAT offset, during every controller cycle.

The method call is legal only when the EtherCAT State is OP.

Syntax

object.MapEtherCATOutput(MotionFlags flags, int offset, string variableName)

Async Syntax

ACSC_WAITBLOCK object.MapEtherCATOutputAsync(MotionFlags flags, int offset, string var

Arguements

flags	Bit-mapped parameter. Currently should be ACSC_NONE.
offset	Internal EtherCAT offset of network output variable. Should be taken from the response to the #ETHERCAT command in the SPiiPlus MMI Application Studio **Communication Terminal** or can be seen via **EtherCAT Configurator.**
variableName	Valid name of ACSPL+ variable, global or standard

Return Value

None

Example

In the following example, network variable of EtherCAT node 0 at offset 26 is being mapped to global variable: **I1**.

```
// Example synchronous call MapEtherCATOutput
double DBuffer = api.GetDBufferIndex();
api.AppendBuffer((ProgramBuffer)DBuffer, "global int IO");
api.CompileBuffer((ProgramBuffer)DBuffer);
api.MapEtherCATOutput(EtherCATFlags.ACSC_NONE, 104, "IO");
```

3.2.5 UnmapEtherCATInputsOutputs

Description

The method resets all previous mapping defined by MapEtherCATInput and "MapEtherCATOutput" on the previous page methods.

The method call is legal only when the EtherCAT State is OP.

Syntax

object.UnmapEtherCATInputsOutputs()

Async Syntax

ACSC_WAITBLOCK object.UnmapEtherCATInputsOutputsAsync()

Arguments

None

Return Value

None

Example

```
// Example synchronous call UnmapEtherCATInputsOutputs
Ch.UnmapEtherCATInputsOutputs();
```

3.2.6 GetEtherCATSlaveIndex

Description

The method retrieves the index of an EtherCAT slave according to provided parameters.

Syntax

object.GetEtherCATSlaveIndex(int vendorID, int productID, int count)

Async Syntax

ACSC_WAITBLOCK object.GetEtherCATSlaveIndexAsync(int vendorID, int productID, int count)

Arguments

vendorID	EtherCAT Vendor ID of the desired slave.
productID	EtherCAT Product ID of the desired slave.
count	Internal count of devices with the same Product and Vendor IDs.

Return Value

Double

```
// Example synchronous call GetEtherCATSlaveIndex
double sIndex = Ch.GetEtherCATSlaveIndex(0x2,0x10243052,0);
```

3.2.7 GetEtherCATSlaveOffsetV2

Description

The method retrieves the offset of a network variable of a specified EtherCAT slave in EtherCAT telegram in a specific network.

Syntax

public double GetEtherCATSlaveOffsetV2(EtherCATFlags flags, string variableName, int slaveIndex)

Async Syntax

public ACSC_WAITBLOCK GetEtherCATSlaveOffsetV2Async(EtherCATFlags flags, string variableName, int slaveIndex)

Arguments

flags	See EtherCAT Flags.
variableName	Name of the EtherCAT network variable.
slaveIndex	Index of the required EtherCAT slave, which can be determined by using the GetEtherCATSlaveIndex method.

Return Value

Double

Example

```
//get the offset of a variable ('DC1') in slave 0:
double slvOffSet = api.GetEtherCATSlaveOffsetV2(EtherCATFlags.ACSC_
ETHERCAT_NETWORK_0, "DC1", 0);
```

Version Support

This method is supported in version 3.13 and later.

3.2.8 GetEtherCATSlaveVendorIDV2

Description

The method retrieves the Vendor ID of a specified EtherCAT slave.

Syntax

public double GetEtherCATSlaveVendorIDV2(EtherCATFlags flags, int slaveIndex)

Async Syntax

public ACSC_WAITBLOCK GetEtherCATSlaveVendorIDV2Async(EtherCATFlags flags, int slaveIndex)

Arguments

flags	See EtherCAT Flags.
slaveIndex	Index of the required EtherCAT slave, which can be determined by using the GetEtherCATSlaveIndex method.

Return Value

Double

Example

```
//get slave's (index 0) vendor id:
double slvVID = api.GetEtherCATSlaveVendorIDV2(EtherCATFlags.ACSC_
ETHERCAT_NETWORK_0, 0);
```

Version Support

This method is supported in version 3.13 and later.

3.2.9 GetEtherCATSlaveProductIDV2

Description

The method retrieves the Product ID of a specified EtherCAT slave.

Syntax

public double GetEtherCATSlaveProductIDV2(EtherCATFlags flags, int slaveIndex)

Async Syntax

public ACSC_WAITBLOCK GetEtherCATSlaveProductIDV2Async(EtherCATFlags flags, int slaveIndex)

Arguments

flags	See EtherCAT Flags.
SlaveIndex	Index of the required EtherCAT slave, which can be determined by using the GetEtherCATSlaveIndex method.

Return Value

Double

Example

```
 //get slave's (index 0) product id:
double slvPID =
   Ch.GetEtherCATSlaveProductIDV2(EtherCATFlags.ACSC_ETHERCAT_NETWORK_0,
0);
```

Version Support

This method is supported in version 3.13 and later.

3.2.10 GetEtherCATSlaveSerialNumber

Description

The method returns the Serial Number of the specified EtherCAT slave in a specific ECAT network.

Syntax

public double GetEtherCATSlaveSerialNumber(EtherCATFlags flags, int slaveIndex)

Async Syntax

public ACSC_WAITBLOCK GetEtherCATSlaveSerialNumberAsync(EtherCATFlags flags, int slaveIndex)

Arguments

flags	See EtherCAT Flags.
SlaveIndex	Index of the required EtherCAT slave, which can be determined by using the GetEtherCATSlaveIndex method.

Return Value

Double

Example

```
//get slave's (index 0) serial number:
double slvSN = api.GetEtherCATSlaveSerialNumber(EtherCATFlags.ACSC_
ETHERCAT_NETWORK_0, 2);
```

Version Support

This method is supported in version 3.13 and later.

3.2.11 GetEtherCATSlaveRevisionV2

Description

The method retrieves the revision of a specified EtherCAT slave.

Syntax

public double GetEtherCATSlaveRevisionV2(EtherCATFlags flags, int slaveIndex)

Async Syntax

public ACSC_WAITBLOCK GetEtherCATSlaveRevisionV2Async(EtherCATFlags flags, int slaveIndex)

Arguments

flags	See EtherCAT Flags.
slaveIndex	Index of the desired EtherCAT slave, which can be determined by using the

Return Value

Double

Example

```
//get slave's (index 0) revision number:
double slvRev = api.GetEtherCATSlaveRevisionV2(EtherCATFlags.ACSC_
ETHERCAT_NETWORK_0, 0);
```

Version Support

This method is supported in version 3.13 and later.

3.2.12 GetEtherCATSlaveType

Description

The method retrieves the type of a specified EtherCAT slave.

Syntax

object.GetEtherCATSlaveType(int vendorID, int productID)

Async Syntax

ACSC_WAITBLOCK object.GetEtherCATSlaveTypeAsync(int vendorID, int productID)

Arguments

vendorID	Vendor ID of the EtherCAT slave.
productID	Product ID of the EtherCAT slave.

Return Value

Double

The type of specified EtherCAT slave. The value can be one of the following:

- 0 – ACS device
- 1 – Non-ACS Servo
- 2 – Non-ACS Stepper
- 3 – Non-ACS I/O
- -1 – Device not found at slave index

Example

```
// Example synchronous call GetEtherCATSlaveType
double type = Ch.GetEtherCATSlaveType(0x2,0x044c2c52);
```

3.2.13 GetEtherCATSlaveStateV2

Description

The method retrieves the EtherCAT state of a specified EtherCAT slave.

Syntax

public double GetEtherCATSlaveStateV2(EtherCATFlags flags, int slaveIndex)

Async Syntax

public ACSC_WAITBLOCK GetEtherCATSlaveStateV2Async(EtherCATFlags flags, int slaveIndex)

Arguments

flags	See EtherCAT Flags.
SlaveIndex	Index of the required EtherCAT slave, which can be determined by using the GetEtherCATSlaveIndex method.

Return Value

Double

Returns the state of specified EtherCAT slave. The value can be one of the following:

- 1 – INIT
- 2 – PREOP
- 4 – SAFEOP
- 8 – OP

> -1 – Device not found at slave index

Example

```
/get slave's (index 0) state:
double slvState = api.GetEtherCATSlaveStateV2(EtherCATFlags.ACSC_
ETHERCAT_NETWORK_0, 0);
```

Version Support

This method is supported in version 3.13 and later.

3.2.14 GetEtherCATSlaveRegister

Description

The function returns value of the ESC Error Counters Registers of specified EtherCAT slave in specific ECAT network.

Syntax

public double GetEtherCATSlaveRegister(EtherCATFlags flags, int slaveIndex, int offset)

Async Syntax

public ACSC_WAITBLOCK GetEtherCATSlaveRegisterAsync(EtherCATFlags flags, int slaveIndex, int offset)

flags	See EtherCAT Flags.
slaveIndex	Index of the required EtherCAT slave, which can be determined by using the GetEtherCATSlaveIndex method.
offset	Register offset in the ESC memory.

Comments

The following table lists supported error counter registers.

Table 3-5. EtherCAT Error Counter Registers

Offset	Name	Description
0x300	Port Error Counter (CRC A)	Error Counted at the Auto-Forwarded (per port). Each register contains two counters: > Invalid Frame Counter: 0x300/2/4/6 > RX Error Counter: 0x301/3/5/7
0x302	Port Error Counter (CRC B)	
0x304	Port Error Counter (CRC C)	

Offset	Name	Description
0x306	Port Error Counter (CRC D)	
0x308	Forwarded RX Error Counter (CRC A/B)	Invalid frame with marking from previous ESC detected (per port).
0x309	Forwarded RX Error Counter	
0x30A	Forwarded RX Error Counter (CRC C/D)	
0x30B	Forwarded RX Error Counter	
0x30C	ECAT Processing Unit Error Counter	Invalid frame passing the EtherCAT Processing Unit (additional checks by processing unit).
0x30D	PDI Error Counter	Physical Errors detected by the PDI.
0x310	Lost Link Counter, Port A (IN)	Link Lost events (per port).
0x311	Lost Link Counter, Port B (OUT)	
0x312	Lost Link Counter, Port C	
0x313	Lost Link Counter, Port D	

The above table contains the offsets and the descriptions of the error counters' registers.

Return Value

ESC Error Counters Registers

Example

```
//get slave's (index 0) register of a specific offset:
double slvOffSet = 0;
double slvREG = api.GetEtherCATSlaveRegister(EtherCATFlags.ACSC_ETHERCAT_
NETWORK_0, 0, (int)slvOffSet);
```

Version Support

This method is supported in version 3.13 and later.

3.2.15 ClearEtherCATSlaveRegister

Description

The method clears the contents of the error counters registers of specified EtherCAT slave in the specific ECAT network.

Syntax

public double GetEtherCATSlaveRegister(EtherCATFlags flags, int slaveIndex, int offset)

Async Syntax

public ACSC_WAITBLOCK GetEtherCATSlaveRegisterAsync(EtherCATFlags flags, int slaveIndex, int offset)

flags	See EtherCAT Flags.
slaveIndex	Index of the required EtherCAT slave, which can be determined by using the GetEtherCATSlaveIndex method.
offset	Register offset in the ESC memory.

Comments

The following table lists supported error counter registers.

Table 3-6. EtherCAT Error Counter Registers

Offset	Name	Description
0x300	Port Error Counter (CRC A)	Error Counted at the Auto-Forwarded (per port). Each register contains two counters: > Invalid Frame Counter: 0x300/2/4/6 > RX Error Counter: 0x301/3/5/7
0x302	Port Error Counter (CRC B)	
0x304	Port Error Counter (CRC C)	
0x306	Port Error Counter (CRC D)	
0x308	Forwarded RX Error Counter (CRC A/B)	Invalid frame with marking from previous ESC detected (per port).
0x309	Forwarded RX Error Counter	
0x30A	Forwarded RX Error Counter (CRC C/D)	

Offset	Name	Description
0x30B	Forwarded RX Error Counter	
0x30C	ECAT Processing Unit Error Counter	Invalid frame passing the EtherCAT Processing Unit (additional checks by processing unit).
0x30D	PDI Error Counter	Physical Errors detected by the PDI.
0x310	Lost Link Counter, Port A (IN)	Link Lost events (per port).
0x311	Lost Link Counter, Port B (OUT)	
0x312	Lost Link Counter, Port C	
0x313	Lost Link Counter, Port D	

The above table contains the offsets and the descriptions of the error counters' registers.

When the offset value is -1, all error counters in all slaves in specific ECAT network are cleared. Otherwise, only the specific register specified by the offset is cleared.

Return Value

ESC Error Counters Registers

Example

```
//reset slave's (index 0) register of a specific offset:
api.ClearEtherCATSlaveRegister(EtherCATFlags.ACSC_ETHERCAT_NETWORK_0, 0,
(int)slvOffSet);
```

Version Support

This method is supported in version 3.13 and later.

3.2.16 GetEtherCATSlavesCount

Description

This method retrieves the number of slaves in the specified EtherCAT network.

Syntax

public double GetEtherCATSlavesCount(EtherCATFlags flags)

Async Syntax

public ACSC_WAITBLOCK GetEtherCATSlavesCountAsync(EtherCATFlags flags)

Arguments

flags	See EtherCAT Flags.

Return Value

Double, indicating the number of slaves in the specified EtherCAT network

Example

```
//get the slaves count in the first network.(amount of connected slaves):
double slvCnt = api.GetEtherCATSlavesCount(EtherCATFlags.ACSC_ETHERCAT_
NETWORK_0);
```

Version Support

This method is supported in version 3.13 and later.

3.3 Service Communication Methods

The Service Communication methods are:

Table 3-7. Service Communication Methods

Method	Description
GetACSHandle	Retrieves the SPiiPlus C Library communication handle
GetNETLibraryVersion	Retrieves the SPiiPlus NET Library version number.
GetCommOptions	Retrieves the communication options.
GetDefaultTimeout	Retrieves default communication timeout.
GetErrorString	Retrieves the explanation of an error code.
GetLibraryVersion	Retrieves the legacy SPiiPlus C Library version number.
GetTimeout	Retrieves communication timeout.
SetIterations	Sets the number of iterations of one transaction.
SetCommOpions	Sets the communication options.
SetTimeout	Sets communication timeout.

3.3.1 GetACSCHandle

Description

The method retrieves the SPiiPlus C Library communication handle.

Syntax

object.GetACSCHandle()

Arguments

None

Return Value

IntPtr

Remarks

The method retrieves the SPiiPlus C Library communication handle. If the method fails, the Error object is filled with the Error Description.

Example

```
// Retrieves communication handle
IntPtr handle = Ch.GetACSCHandle();
```

3.3.2 GetNETLibraryVersion

Description

The method retrieves the SPiiPlus NET Library version number.

Syntax

object.GetNETLibraryVersion()

Arguments

None

Return Value

UInt32.

The method retrieves the SPiiPlus NET Library version number.

Remarks

The SPiiPlus NET Library version consists of four (or less) numbers separated by points:

#.#.#.#. The binary version number is represented by 32-bit unsigned integer value. Each byte of this value specifies one number in the following order: high byte of high word – first number, low byte of high word – second number, high byte of low word – third number and low byte of low word – forth number. For example the version "2.10" has the following binary representation (hexadecimal format): 0x020A0000.

The first two numbers in the string form are obligatory. Any release version of the library consists of two numbers. The third and fourth numbers specify an alpha or beta version, special or private build, etc.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Retrieves the SPiiPlus .NET Library
uint version = Ch.GetNETLibraryVersion();
```

3.3.3 GetCommOptions

Description

The method retrieves the communication options.

Syntax

object.GetCommOptions()

None

Return Value

CommOptions.

The method retrieves the current communication options.

Remarks

To set the communication options call SetCommOptions.

The Return Value is bit-mapped to represent the current communication options. Currently only the following option flag is supported:

ACSC_COMM_USE_CHECKSUM - communication mode when each command sends to the controller with checksum and the controller responds with checksum.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Retrieves the current communication options
CommOptions options = Ch.GetCommOptions();
```

3.3.4 GetDefaultTimeout

Description

The method retrieves default communication timeout.

Syntax

object.GetDefaultTimeout()

Arguments

None

Return Value

Int32.

Remarks

The value of the default timeout depends on the type of the established communication channel. Timeout depends also on the baud rate value for serial communication.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Retrieves default communication timeout
int timeout = Ch.GetDefaultTimeout();
```

3.3.5 GetErrorString

Description

The method retrieves the explanation of an error code.

Syntax

object.GetErrorString(ErrorCodes

Arguments

ErrorCode	An error code returned by the following methods: *GetMotorError GetMotionError GetProgramError*

Return Value

String

The method retrieves the string that contains the text explanation of the error code returned by the *GetMotorError*, *GetMotionError*, and *GetProgramError* methods.

Remarks

If the error relates to SPiiPlus NET Library, the method returns immediately with the text explanation. If the error relates to the controller, the method receives the text explanation from the controller.

If the method fails, the Error object is filled with the Error Description.
Example

```
// The method retrieves the explanation of an error code 3260.
string s = Ch.GetErrorString(3260);
```

3.3.6 GetLibraryVersion

Description

The method retrieves the SPiiPlus C Library version number.

Syntax

object.GetLibraryVersion()

Arguments

None

Return Value

Uint32.

The method retrieves the legacy SPiiPlus C Library version number.

Remarks

The SPiiPlus C Library version consists of four (or less) numbers separated by points: #.#.#.#. The binary version number is represented by 32-bit unsigned integer value. Each byte of this value specifies one number in the following order: high byte of high word – first number, low byte of high word – second number, high byte of low word – third number and low byte of low word – forth number. For example the version “2.10” has the following binary representation (hexadecimal format): 0x020A0000.

The first two numbers in the string form are obligatory. Any release version of the library consists of two numbers. The third and fourth numbers specify an alpha or beta version, special or private build, etc.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Retrieves the SPiiPlus C Library version number
uint libVersion = Ch.GetLibraryVersion();
```

3.3.7 GetTimeout

Description

The method retrieves communication timeout.

Syntax

object.GetTimeout()

Arguments

None

Return Value

Int32.

The method retrieves communication timeout.

Remarks

If the method succeeds, the return value is the current timeout value in milliseconds. If the method fails, the Error object is filled with the Error Description.

Example

```
// Retrieves communication timeout
int timeout = Ch.GetTimeout();
```

3.3.8 SetIterations

Description

The method sets the number of iterations in one transaction.

Syntax

object.SetIterations(int iterations)

Arguments

iterations	Number of iterations

Return Value

None

Remarks

If, after the transmission of command to the controller, the controller response is not received during the predefined time, the library repeats the command transmission. The number of those iterations is defined by **iterations** parameter for each communication channel independently.

Most of the SPiiPlus NET methods perform communication with the controller by transactions (i.e. they send commands and wait for responses) that are based on the *Transaction* method.

Therefore, the changing of the number of iterations can have an influence on the behavior of the user application.

The default number of iterations for all communication channels is 2. If the method fails, the Error object is filled with the Error Description.

Example

```
// Sets the number of iterations in one transaction
// number of iterations for all communication channels is 2
Ch.SetIterations(2);
```

3.3.9 SetCommOptions

Description

The method sets the communication options.

Syntax

object.SetCommOptions(CommOptions options)

Arguments

options	Communication options to be set. Bit-mapped parameter that can include one of the following flags: > **ACSC_COMM_USE_CHECKSUM**: the communication mode used when each Command is sent to the controller with checksum and the controller also responds with checksum. > **ACSC_COMM_AUTORECOVER_HW_ERROR**: when a hardware error is detected in the communication channel and this bit is set, the library automatically repeats the transaction, without counting iterations.

Return Value

None

Remarks

The method sets the communication options. To get current communication option, call *GetCommOptions*.

To add some communication options to the current configuration, modify **options** that have been filled in by a call to *GetCommOptions*. This ensures that the other communication options will have same values.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Example sets the mode with checksum
CommOptions options= Ch.GetCommOptions();
Ch.SetCommOptions( options | CommOptions.ACSC_COMM_USE_CHECKSUM );
```

3.3.10 SetTimeout

Description

The method sets the communication timeout.

Syntax

object.SetTimeout(int timeout)

Arguments

timeout	Maximum waiting time in milliseconds.

Return Value

None

Remarks

The method sets the communication timeout.

All of the subsequent waiting calls of the methods will wait for the controller response timeout in milliseconds. If the controller does not respond to the sent command during this time, SPiiPlus NET methods will throw ACSException with the error code assigned to ACSC_TIMEOUT.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Sets the communication timeout of 2 seconds
Ch.SetTimeout(2000);
```

3.4 ACSPL+ Program Management Methods

The ACSPL+ Program Management methods are:

Table 3-8. ACSPL+ Program Management Methods

Method	Description
AppendBuffer	Appends one or more ACSPL+ lines to the program in the specified program buffer.
ClearBuffer	Deletes the specified ACSPL+ program lines in the specified program buffer.
CompileBuffer	Compiles ACSPL+ program in the specified program buffer(s).
LoadBuffer	Clears the specified program buffer and then loads ACSPL+ program to this buffer.
LoadBuffersFromFile	Opens a file that contains one or more ACSPL+ programs allocated to several buffers and download the programs to the corresponding buffers.
RunBuffer	Starts up ACSPL+ program in the specified program buffer.
StopBuffer	Stops ACSPL+ program in the specified program buffer(s).
SuspendBuffer	Suspends ACSPL+ program in the specified program buffer(s).
UploadBuffer	Uploads ACSPL+ program from the specified program buffer.

3.4.1 AppendBuffer

Description

The method appends one or more ACSPL+ lines to the program in the specified buffer.

Syntax

object.AppendBuffer(ProgramBuffer buffer, string text)

Async Syntax

ACSC_WAITBLOCK object.AppendBufferAsync(ProgramBuffer buffer, string text)

Arguments

Buffer	Number of a program buffer in the controller.
text	String containing an ACSPL+ program(s).

Return Value

None

Remarks

The method appends one or more ACSPL+ lines to the program in the specified buffer. If the buffer already contains any program, the new text is appended to the end of the existing program.

No compilation or syntax check is provided during downloading. In fact, any text, not only a correct program, can be inserted into a buffer. In order to compile the program and check its accuracy, the compile command must be executed after downloading.

If the method fails, the Error object is filled with the Error Description.

Example

```
string program = "enable x; jog x; stop\n";
// The method appends one ACSPL+ line to buffer 0
Ch.AppendBuffer(ProgramBuffer.ACSC_BUFFER_0, program);
```

3.4.2 ClearBuffer

Description

Deletes the specified ACSPL+ program lines in the specified program buffer.

Syntax

object.ClearBuffer(ProgramBuffer buffer, int fromLine, int toLine)

Async Syntax

ACSC_WAITBLOCK object.ClearBufferAsync(ProgramBuffer buffer, int fromLine, int toLine)

Arguments

buffer	Buffer number
FromLine, ToLine	These *Arguments* specify a range of lines to be deleted. **fromLine** starts from 1. If **toLine** is larger then the total number of lines in the specified program buffer, the range includes the last program line.

Return Value

None

Remarks

The method deletes the specified ACSPL+ program lines in the specified program buffer. If the method fails, the Error object is filled with the Error Description.

Example

```
string program = "enable x; jog x; stop\n";
// The method appends one ACSPL+ line to buffer 0
Ch.AppendBuffer(ProgramBuffer.ACSC_BUFFER_0, program);
// Get program of buffer 0
string buf = Ch.UploadBuffer(ProgramBuffer.ACSC_BUFFER_0);
// Delete buffer 0 from line 1 to 1000
Ch.ClearBuffer(ProgramBuffer.ACSC_BUFFER_0, 1, 1000);
// Get program of buffer 0
buf = Ch.UploadBuffer(ProgramBuffer.ACSC_BUFFER_0);
```

3.4.3 CompileBuffer

Description

The method compiles the ACSPL+ program in the specified program buffer(s).

Syntax

object.CompileBuffer(ProgramBufferbuffer)

Async Syntax

ACSC_WAITBLOCK object.CompileBufferAsync(ProgramBufferbuffer)

Arguments

buffer	Buffer number You can use **ACSC_NONE** instead of the buffer number to compile all programs in all buffers.

Return Value

None

Remarks

The method compiles an ACSPL+ program in the specified program buffer or all programs in all **buffers** if buffer is **ACSC_NONE**.

> If attempting to compile the D-Buffer, all other buffers will be stopped and put in a non-compiled state.

The method succeeds if the compile command was transmitted successfully to the controller, i.e., the communication channel is OK and the specified buffer was not running. However, this does not mean that the compile operation was completed successfully.

In order to get information about the results of the compile operation, use *ReadVariable* to read **PERR**, which contains the most recent error that occurred in each buffer. If **PERR** is zero. the buffer was compiled successfully.

Otherwise, **PERR** contains the error that occurred during the compilation. The method waits for the controller response. If the method fails, the Error object is filled with the Error Description.

Example

```
string program = "!Compiled buffer; stop\n";
// Appends line to buffer 0
Ch.AppendBuffer(ProgramBuffer.ACSC_BUFFER_0, program);
// The method compiles ACSPL+ program in buffer 0
Ch.CompileBuffer(ProgramBuffer.ACSC_BUFFER_0);
```

3.4.4 LoadBuffer

Description

The method clears the specified program buffer and then loads ACSPL+ program to this buffer.

Syntax

object.LoadBuffer(ProgramBuffer buffer, string text)

Async Syntax

ACSC_WAITBLOCK object.LoadBufferAsync(ProgramBuffer buffer, string text)

Arguments

buffer	Number of a program buffer in the controller.
text	String containing an ACSPL+ program(s).

Return Value

None

Remarks

The method clears the specified program buffer and then loads ACSPL+ program to this buffer.

No compilation or Syntax check is provided during downloading. Any text, not only a correct program, can be inserted into a buffer. In order to compile the program and check its accuracy, the compile command must be executed after downloading.

If the method fails, the Error object is filled with the Error Description.

Example

```
string program = "!This is a test ACSPL + program; Stop\n";
// Load a line to buffer 0
Ch.LoadBuffer(ProgramBuffer.ACSC_BUFFER_0, program);
```

3.4.5 LoadBuffersFromFile

Description

The method opens a file that contains one or more ACSPL+ programs allocated to several buffers and download the programs to the corresponding buffers.

Syntax

object.LoadBuffersFromFile(string filename)

Async Syntax

ACSC_WAITBLOCK object.LoadBuffersFromFileAsync(string filename)

Arguments

filename	Name and path of the file to be loaded.

Return Value

None

Remarks

The method analyzes the file, determines which program buffers should be loaded, clears them and then loads ACSPL+ programs to those buffers.

The method can be called only synchronously.

SPiiPlus software tools save ACSPL+ programs in the following format:

```
# Header: Date, Firmware version, etc.
#Buf1 (buffer number) ACSPL+ program of Buf1
#Buf2 (buffer number) ACSPL+ program of Buf2
#Buf3 (buffer number) ACSPL+ program of Buf3 etc.
```

The number of buffers in file may change from 1 to 10, without any default order.

No compilation or Syntax check is provided during downloading. Any text, not only a correct program, can be inserted into a buffer. In order to compile the program and check its accuracy, the compile command must be executed after downloading.

If the method fails, the Error object is filled with the Error Description.

Example

```
String file = "C:\\ACSPLFile.txt";
// Opens a file to load
Ch.LoadBuffersFromFile(file);
```

3.4.6 RunBuffer

Description

The method starts up ACSPL+ program in the specified buffer.

Syntax

object.RunBuffer(ProgramBuffer buffer, [string label])

Async Syntax

ACSC_WAITBLOCK object.RunBufferAsync(ProgramBuffer buffer, [string label])

Arguments

buffer	Number of a program buffer in the controller.
Label	Label in the program from which the execution starts. If NULL ("" - the default) is specified instead of a pointer, the execution starts from the first line.

Return Value

None

Remarks

The method starts up ACSPL+ program in the specified buffer. The execution starts from the specified label, or from the first line if the label is not specified.

If the program was not compiled before, the method first compiles the program and then starts it. If an error was encountered during compilation, the program does not start.

If the program was suspended by the *SuspendBuffer* method, the method resumes the program execution from the point where the program was suspended.

The method waits for the controller response.

The controller response indicates that the program in the specified buffer was started successfully. The method does not wait for the program end. To wait for the program end, use the WaitProgramEnd method.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Appends and runs buffers 0 to 8
for (int index = 0 ; index < 8 ; index++)
{
```

```
  Ch.AppendBuffer((ProgramBuffer)index,"ST:");
  Ch.AppendBuffer((ProgramBuffer)index,"!an empty loop");
  Ch.AppendBuffer((ProgramBuffer)index,"goto ST");
  Ch.RunBuffer((ProgramBuffer)index,null);
}
```

3.4.7 StopBuffer

Description

The method stops the execution of ACSPL+ program in the specified buffer(s).

Syntax

object.StopBuffer(ProgramBuffer buffer)

Async Syntax

ACSC_WAITBLOCK object.StopBufferAsync(ProgramBuffer buffer)

Arguments

buffer	Buffer number. To stop the execution of all buffers, use ACSC_NONE instead of the buffer number.

Return Value

None

Remarks

The method stops ACSPL+ program execution in the specified buffer or in all buffers if **Buffer** is **ACSC_NONE**.

The method has no effect if the program in the specified buffer is not running. If the method fails, the Error object is filled with the Error Description.

Example

```
// Appends and runs buffers 0 to 8
for (int index = 0; index < 8; index++)
{
       // Append line to buffer
        Ch.AppendBuffer((ProgramBuffer)index, "ST:; !Empty buffer");
        // Append line to buffer
        Ch.AppendBuffer((ProgramBuffer)index, "goto ST;");
        // Append line to buffer
        Ch.AppendBuffer((ProgramBuffer)index, "stop");
        // Run buffer
        Ch.RunBuffer((ProgramBuffer)index, null);
        // Stop running
        Ch.StopBuffer((ProgramBuffer)index);
```

3.4.8 SuspendBuffer

Description

The method suspends the execution of ACSPL+ program in the specified program buffer(s).

Syntax

object.SuspendBuffer(ProgramBuffer buffer)

Async Syntax

ACSC_WAITBLOCK object.SuspendBufferAsync(ProgramBuffer buffer)

Arguments

buffer	Buffer number. To suspend the execution of all buffers, use **ACSC_NONE** instead of the buffer number.

Return Value

None

Remarks

The method suspends ACSPL+ program execution in the specified buffer or in all buffers if **buffer** is **ACSC_NONE.** The method has no effect if the program in the specified buffer is not running.

To resume execution of the program in the specified buffer, call the *RunBuffer* method. If the method fails, the Error object is filled with the Error Description.

Example

```
// Appends and runs buffers 0 to 8
for (int index = 0 ; index < 8 ; index++)
{
  Ch.AppendBuffer((ProgramBuffer)index,"ST:");
  Ch.AppendBuffer((ProgramBuffer)index,"!an empty loop");
  Ch.AppendBuffer((ProgramBuffer)index,"goto ST");
  Ch.RunBuffer((ProgramBuffer)index,null);
}
// Suspend all buffers
Ch.SuspendBuffer(ProgramBuffer.ACSC_NONE);
```

3.4.9 UploadBuffer

Description

The method uploads ACSPL+ program from the specified program buffer.

Syntax

object.UploadBuffer(ProgramBuffer buffer)

Arguments

Buffer	Number of a program buffer in the controller.

Return Value

String

Remarks

The method uploads ACSPL+ program from the specified program buffer. If the method fails, the Error object is filled with the Error Description.

Example

```
String program = "!This is an empty buffer\n";
// Appendsprogram to buffer 0
Ch.AppendBuffer(ProgramBuffer.ACSC_BUFFER_0, program);
// Get program of buffer 0
String buf = Ch.UploadBuffer(ProgramBuffer.ACSC_BUFFER_0);
```

3.5 Read and Write Variables Methods

The Read and Write Variable methods are:

Table 3-9. Read and Write Variables Methods

Method	Description
ReadVariable	Reads variable from a controller and returns it in the form of object
ReadVariableAsScalar	Reads variable from a controller and returns it as scalar object
ReadVariableAsVector	Reads variable from a controller and returns it in the form object that contains one-dimension array
ReadVariableAsMatrix	Reads variable from a controller and returns it in the form object that contains two-dimensions array
WriteVariable	Writes to controller variable(s)

3.5.1 ReadVariable

Description

The method reads a variable from the controller.

Syntax

object.ReadVariable(string variable, [ProgramBuffer nBuf], [int from1], [int to1], [int from2], [int to2])

Async Syntax

ACSC_WAITBLOCK object.ReadVariableAsync(string variable, [ProgramBuffer nBuf], [int from1], [int to1], [int from2], [int to2])

Arguments

variable	Name of the controller variable
nBuf	Number of program buffer for local variable or ProgramBuffer.ACSC_NONE for global and ACSPL+ variables.
from1, to1	Index range (first dimension) starting from zero. Default value: Api.ACSC_NONE.
from2, to2	Index range (second dimension) starting from zero. Default value: Api.ACSC_NONE.

Return Value

Object.

The type of return value is real or integer, depending on the type of controller variable.

The return value can be scalar, vector (one-dimensional array) or matrix (two-dimensional array) depending on data that the controller retrieves.

Remarks

The method reads scalar variables, vectors, rows of matrix, and columns of matrix or matrices from a controller and retrieves data as object. The resultant data are initialized depending on what data is received from the controller: scalar, vector or matrix.

If you want the return value to be in some specified type, use *ReadVariableAsScalar*, *ReadVariableAsVector* or *ReadVariableAsMatrix* methods.

The variable can be an ACSPL+ controller variable, a user global variable, or a user local variable. ACSPL+ and user global variables have global scope; therefore **nBuf** must be omitted or specified as **ProgramBuffer.ACSC_NONE** (-1) for these classes of variables.

User local variable exists only within a buffer. The buffer number must be specified for user local variable.

The index parameters can be specified as follows:

> To read the whole variable
> To read one element of the variable
> To read several elements of the variable

The following explains how to specify indexes in several typical cases.

1. Reading whole variable

   The **from1, to1, from2** and **to2** indexes should be omitted or specified as **Api.ACSC_NONE**. The method returns all elements of **Variable** as object initialized as:

   > Scalar, if the Variable is scalar
   > Vector, if the Variable is a one-dimensional array

> Matrix, if the Variable is a two-dimensional array

2. Reading one element from vector (one-dimensional array)

   The **from1** and **to1** should be equal and specify the index of the element. **from2** and **to2** should be omitted or specified as **Api.ACSC_NONE**. The method returns the element value as object, initialized as scalar.

3. Reading one element from matrix (two-dimensional array)

   from1 should equal **to1** and specify row number. The **from2** and **to2** should be equal and specify column number. The method returns the element value as object, initialized as scalar method

4. Reading a sub-vector (more than one vector element)

   from1 and **to1** should specify the sub-vector index range, where **from1** should be less than **to1**. **from2** and **to2** should be omitted or specified as **Api.ACSC_NONE**. The method returns data as object, initialized as a vector with the number of elements equal to **to1– from1** +1.

5. Reading a row or part of a row from a matrix

   from1 should equal to1and specify the row number. The **from2** and **to2** should specify the element range within the specified row, where **from2** should be less than **to2**. The method returns data as object, initialized as a vector with the number of elements equal **to2 – from2** +1.

6. Reading a column or part of a column from a matrix

   from1 and **to1** should specify the range of rows within the specified column, where **from1** should be less than **to1**. **from2** and **to2** should be equal and specify the column number. The method returns data as object, initialized as a vector with the number of elements equal to **to1–from1**+1.

7. Reading sub-matrix from matrix

   from1 and **to1** should specify the range of rows, where **from1** should be less then **to1**. **from2** and **to2** should specify columns range, where **from2** should be less than **to2**. The method returns data as object, initialized as a matrix with the number of rows equal to **to1– from1** +1 and the number of columns equal to to**2 – from2** +1.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Read variable
object var;
//Reading whole variable, if the variable is scalar:
//Parameters: "BAUD" - controller variable name
//(optional:)ProgramBuffer.ACSC_NONE - ACSPL+ variable
var = Ch.ReadVariable("BAUD");
//or:
var = Ch.ReadVariable("BAUD", ProgramBuffer.ACSC_NONE);
//Now var is a scalar value
//Reading whole variable, if the variable is
//one-dimensional array:
```

```
//Parameters: "VEL" - controller variable name
//(optional:)ProgramBuffer.ACSC_NONE - ACSPL+ variable
var = Ch.ReadVariable("VEL");
//or:
var = Ch.ReadVariable("VEL", ProgramBuffer.ACSC_NONE);
//Now var is a vector of size 8
//Reading whole variable, if the variable is
//two-dimensional array:
//Assumed that MyMatrix variable is declared as global
//MyMatrix(3)(4) in the controller:
//Parameters: "MyMatrix" – user variable name
//(optional:)ProgramBuffer.ACSC_NONE - global variable
var = Ch.ReadVariable("MyMatrix");
//or:
var = Ch.ReadVariable("MyMatrix", ProgramBuffer.ACSC_NONE);
//Now var is a matrix of size 3x4
//Reading one element from vector (one-dimensional array):
//Reading one element with index 3:
//Parameters: "VEL" - controller variable name
//ProgramBuffer.ACSC_NONE - ACSPL+ variable
//elementIndex - from1, to1
int elementIndex = 3;
var = Ch.ReadVariable("VEL", ProgramBuffer.ACSC_NONE,
    elementIndex, elementIndex);
//Now var is a scalar value
//Reading one element from matrix (two-dimensional array):
//Reading one element of matrix from row 1, column 2:
//Assumed that MyMatrix variable is declared as global
//MyMatrix(3)(4) in the controller:
//Parameters: "MyMatrix" – user variable name

//ProgramBuffer.ACSC_NONE - global variable
//elementRow - from1, to1
//elementColumn – from2, to2
int elementRow = 1;
int elementColumn = 2;
var = Ch.ReadVariable("MyMatrix", ProgramBuffer.ACSC_NONE,
    elementRow, elementRow, elementColumn, elementColumn);
//Now var is a scalar value
//Reading sub-vector (more then one element) from vector:
//Reading sub-vector with indexes from 1 to 5:
//Parameters: "VEL" - controller variable name
//ProgramBuffer.ACSC_NONE - ACSPL+ variable
//fromIndex - from1
//toIndex – to1
int fromIndex = 1;
int toIndex = 5;
var = Ch.ReadVariable("VEL", ProgramBuffer.ACSC_NONE,
    fromIndex, toIndex);
```

```
//Now var is a vector of size 5
//Reading row or part of row from matrix:
//Reading a part of row 1 from element with
//index 0 till element with index 2:
//Assumed that MyMatrix variable is declared as global
//MyMatrix(3)(4)
//in the controller:
//Parameters: "MyMatrix" - user variable name
//ProgramBuffer.ACSC_NONE - global variable
//rowIndex - from1, to1
//fromColumnIndex – from2
//toColumnIndex – to2
int rowIndex = 1;
int fromColumnIndex = 0;
int toColumnIndex = 2;
var = Ch.ReadVariable("MyMatrix", ProgramBuffer.ACSC_NONE,
    rowIndex, rowIndex, fromColumnIndex, toColumnIndex);
//Now var is a vector of size 3
//Reading column or part of column from matrix:
//Reading a part of column 1 from element with index 0 till
//element with index 1:
//Assumed that MyMatrix variable is declared as global
//MyMatrix(3)(4) in the controller:
//Parameters: "MyMatrix" - user variable name
//ProgramBuffer.ACSC_NONE - global variable
//fromRowIndex - from1
//toRowIndex – to1
//columnIndex – from2, to2
int fromRowIndex = 0;
int toRowIndex = 1;
int columnIndex = 1;
var = Ch.ReadVariable("MyMatrix", ProgramBuffer.ACSC_NONE,
    fromRowIndex, toRowIndex, columnIndex, columnIndex);
//Now var is a vector of size 2
//Reading sub-matrix from matrix:
//Reading sub-matrix with rows from 0 to 1 and colomns from
//0 to 2:
//Assumed that MyMatrix variable is declared as global
//MyMatrix(3)(4) in the controller:
//Parameters: "MyMatrix" - user variable name
//ProgramBuffer.ACSC_NONE - global variable
//fromRowIndex - from1
//toRowIndex – to1
//fromColumnIndex – from2
//toColumnIndex – to2
fromRowIndex = 0;
toRowIndex = 1;
fromColumnIndex = 0;
toColumnIndex = 2;
```

```
var = Ch.ReadVariable("MyMatrix", ProgramBuffer.ACSC_NONE,
    fromRowIndex, toRowIndex, fromColumnIndex,
    toColumnIndex);
//Now var is a matrix 2x3
```

3.5.2 ReadVariableAsScalar

Description

The method reads the variable from a controller and returns as a scalar.

Syntax

object.ReadVariableAsScalar (string variable, ProgramBuffer nBuf, [int ind1], [int ind2])

Async Syntax

ACSC_WAITBLOCK object.ReadVariableAsScalarAsync(string variable, ProgramBuffer nBuf, [int ind1], [int ind2])

Arguments

variable	Name of the controller variable
nBuf	Number of program buffer for local variable or **ProgramBuffer.ACSC_NONE** for user global and ACSPL+ variables.
ind1	Index of first dimension (row number) starting from zero. Default value: **Api.ACSC_NONE**.
ind2	Index of second dimension (column number) starting from zero. Default value: **Api.ACSC_NONE**.

Return Value

Object.

The return value is scalar, that is, the type of Return Value is real or integer, corresponding to the type of controller variable.

Remarks

Reads a scalar variable, one element of a vector or one element of a matrix from a controller and returns the value as scalar.

If you require a method Return Value as one vector element, use the *ReadVariableAsVector* method. If you require a method Return Value as one matrix element, use the *ReadVariableAsMatrix* method.

The variable can be an ACSPL+ variable, a user-defined global variable, or a user-defined local variable.

ACSPL+ and user global variables have global scope; therefore **nBuf** must be **ProgramBuffer.ACSC_NONE** (–1) for these classes of variables.

User-defined local variables exist only within a buffer. The buffer number must be specified for user-defined local variables.

The following explains how to read indexes in several typical cases.

1. Reading a scalar variable:

   The **ind1** and **ind2** indexes should be omitted or specified as **Api.ACSC_NONE**.

2. Reading one element from a vector (one-dimensional array):

   ind1 should specify index of the element in the vector. **ind2** should be omitted or specified as **Api.ACSC_NONE**.

3. Reading one element from a matrix (two-dimensional array):

   ind1 should specify a row number and **ind2** should specify a column number of the element in the matrix.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Read variable as scalar
//var in the next examples is returned as scalar object var;
//Reading scalar variable:
//Parameters: "BAUD" - controller variable name
//ProgramBuffer.ACSC_NONE - ACSPL+ variable
object var;
var = Ch.ReadVariableAsScalar("BAUD",
    ProgramBuffer.ACSC_NONE);
//Reading one element from vector (one-dimensional array):
//Reading one element with index 2:
//Parameters: "VEL" - controller variable name
//ProgramBuffer.ACSC_NONE - ACSPL+ variable
//index – ind1
int index = 2;
var = Ch.ReadVariableAsScalar("VEL",
    ProgramBuffer.ACSC_NONE, index);
//Reading one element from matrix (two-dimensional array):
//Reading one element of matrix from row 1, column 2:
//Assumed that MyMatrix variable is declared as global
//MyMatrix(3)(4)
//in the controller:
//Parameters: "MyMatrix” – user variable name
//ProgramBuffer.ACSC_NONE - global variable
//rowIndex – ind1
//columnIndex – ind2
int rowIndex = 1;
int columnIndex = 2;
var = Ch.ReadVariableAsScalar("MyMatrix",
    ProgramBuffer.ACSC_NONE, rowIndex, columnIndex);
```

3.5.3 ReadVariableAsVector

Description

The method reads a variable from a controller and returns it as vector.

Syntax

object.ReadVariableAsVector(string variable, [ProgramBuffer nBuf], [int from1], [int to1], [int from2], [int to2])

Async Syntax

ACSC_WAITBLOCK object.ReadVariableAsVectorAsync(string variable, [ProgramBuffer nBuf], [int from1], [int to1], [int from2], [int to2])

Arguments

variable	Name of the controller variable
nBuf	Number of program buffer for local variable or **ProgramBuffer.ACSC_NONE** for global and ACSPL+ variables.
from1, to1	Index range (first dimension) starting from zero. Default value: **Api.ACSC_NONE**.
from2, to2	Index range (second dimension) starting from zero. Default value: **Api.ACSC_NONE**.

Return Value

Object.

The return value is a one-dimensional array (vector). The type of return value is real or integer, corresponding to the type of controller variable.

Remarks

If you want a return value as scalar or matrix use *ReadVariableAsScalar* or *ReadVariableAsMatrix* methods correspondingly.

The variable can be an ACSPL+ variable, a user global variable, or a user local variable. ACSPL+ and user global variables have global scope; therefore **nBuf** must be omitted or specified as **ProgramBuffer.ACSC_NONE** (–1) for these classes of variables.

User local variables exist only within a buffer. The buffer number must be specified for user local variable.

The index parameters can be specified in many different ways to read whole variable, one element of the variable or several elements of the variable. The following explains how to specify indexes in several typical cases.

1. Reading whole variable:

   The **from1, to1**, **from2** and **to2** indexes should be omitted or specified as **Api.ACSC_NONE**.

The method returns all elements of **variable** as object initialized as: Vector of size 1, if the **variable** is scalar

Vector of size N, if the **variable** is vector of size N

If the **variable** is matrix, the method will return an error.

2. Reading sub-vector from vector:

   The **from1** and **to1** should specify sub-vector index range, from1 should be less than or equal to **to1**. The **from2** and **to2** should be omitted or specified as **Api.ACSC_NONE**. The methods returns data as vector with number of elements equals to **to1- from1** +1.

3. Reading a row or part of a row from matrix:

   The **from1** and **to1** should be equal and specify row number. The **from2** and **to2** should specify elements range within the specified row, **from2** should be less than or equal to **to2**. The methods returns data as vector with number of elements equals to **to2 - from2** +1.

4. Reading a column or part of a column from matrix:

   The **from1** and **to1** should specify rows range within the specified column, from1 should be less than or equal to **to1**. The **from2** and **to2** should be equal and specify column number. The methods returns data as vector with number of elements equals to **to1- from1** +1.

If the method fails, the Error object is filled with the Error Description.

Example

```
 // Read variable as vector
 object var;
 //var in the next Examples is returned as vector object var;
 //Reading whole variable, if the variable is scalar:
 //Parameters: "BAUD" - controller variable name
 //(optional:)ProgramBuffer.ACSC_NONE - ACSPL+ variable
 var = Ch.ReadVariableAsVector("BAUD");
 //or:
 var = Ch.ReadVariableAsVector("BAUD",
     ProgramBuffer.ACSC_NONE);
 //now var is a vector of size 1
 //Reading whole variable, if the variable is one-dimensional
 //array:
 //Parameters: "VEL" - controller variable name
 //(optional:)ProgramBuffer.ACSC_NONE - ACSPL+ variable
var = Ch.ReadVariableAsVector("VEL");
 //or:
 var = Ch.ReadVariableAsVector("VEL",
     ProgramBuffer.ACSC_NONE);
 //now var is a vector of size 8
 //Reading sub-vector from vector:
 //Reading sub-vector with indexes from 1 to 5:
 //Parameters: "VEL" - controller variable nam
 //ProgramBuffer.ACSC_NONE - ACSPL+ variable
 //fromIndex – from1
 //toIndex – to1
```

```
int fromIndex = 1;
int toIndex = 5;
var = Ch.ReadVariableAsVector("VEL",
    ProgramBuffer.ACSC_NONE, fromIndex, toIndex);
//now var is a vector of size 5
//Reading row or part of row from matrix:
//Reading a part of row 1 from element with index 0 till
//element with index 2
//Assumed that MyMatrix variable is declared as global
//MyMatrix(3)(4) in the controller:
//Parameters: "MyMatrix" – user variable name
//ProgramBuffer.ACSC_NONE - global variable
//rowIndex – from1, to1
//fromColumnIndex – from2
//toColumnIndex – to2
int rowIndex = 1;
int fromColumnIndex = 0;
int toColumnIndex = 2;
var = Ch.ReadVariableAsVector("MyMatrix",
    ProgramBuffer.ACSC_NONE, rowIndex,
    rowIndex, fromColumnIndex, toColumnIndex);
//Now var is a vector of size 3
//Reading column or part of column from matrix:
//Reading a part of colomn 1 from element with index 0 till
//element with index 1
//Assumed that MyMatrix variable is declared as global
//MyMatrix(3)(4) in the controller:
//Parameters: "MyMatrix" – user variable name
//ProgramBuffer.ACSC_NONE - global variable
//fromRowIndex – from1
//toRowIndex – to1
//columnIndex – from2, to2
int fromRowIndex = 0;
int toRowIndex = 1;
int columnIndex = 1;
var = Ch.ReadVariableAsVector("MyMatrix",
    ProgramBuffer.ACSC_NONE, fromRowIndex, toRowIndex,
    columnIndex, columnIndex);
//Now var is a vector of size 2
```

3.5.4 ReadVariableAsMatrix

Description

The method reads the variable from a controller and returns it as matrix.

Syntax

object.ReadVariableAsMatrix(string variable, [ProgramBuffer nBuf], [int from1], [intto1], [int from2], [int to2])

Async Syntax

ACSC_WAITBLOCK object.ReadVariableAsMatrixAsync(string variable, [ProgramBuffer nBuf], [int from1], [int to1], [int from2], [int to2])

Arguments

variable	Name of the controller variable
nBuf	Number of program buffer for local variable or **ProgramBuffer.ACSC_NONE** for global and ACSPL+ variables.
from1, to1	Index range (first dimension) starting from zero. Default value: **Api.ACSC_NONE**.
from2, to2	Index range (second dimension) starting from zero. Default value: **Api.ACSC_NONE**.

Return Value

Object.

The return value is a two-dimensional array (matrix). The type of return value is real or integer, corresponding to the type of controller variable.

Remarks

If you want a Return Value as scalar or vector use *ReadVariableAsScalar* or *ReadVariableAsVector* methods correspondingly.

The variable can be an ACSPL+ controller variable, a user global variable, or a user local variable.

ACSPL+ variables and user global variables have a global scope; therefore **nBuf** must be omitted or specified as **ProgramBuffer.ACSC_NONE** (–1) for these classes of variables.

User local variables exist only within a buffer. The buffer number must be specified for user local variable.

The index parameters can be specified in many different ways to read whole variable, one element of the variable or several elements of the variable. The following explains how to specify indexes in several typical cases.

- Reading a whole variable

  from1, **to1**, **from2** and **to2** indexes should be omitted or specified as **Api.ACSC_NONE**. The method returns all elements of **variable** as object initialized as follows:

  Matrix 1x1 , if the **variable** is scalar

  Matrix 1xN, if the **variable** is vector of size N Matrix NxM, if the **variable** matrix NxM

- Reading a sub-vector from a vector

  from1 and **to1** should specify the sub-vector index range, **from1** should be less than or equal to **to1**. **from2** and **to2** should be omitted or specified as **Api.ACSC_NONE**. The method returns data as object, initialized as a matrix of size (**to1–from1** +1)x1.

- Reading a sub-matrix from a matrix

from1 and **to1** should specify a range of rows, where **from1** should be less than or equal to **to1**. **from2** and **to2** should specify a range of columns , where **from2** should be less than or equal to **to2**. The method returns data as object, initialized as matrix with the number of rows equal to **to1–from1** +1 and the number of columns equals **to2–from2** +1.

If the method fails, the Error object is filled with the Error Description.

Example

```
//Read variable as matrix
object var;
//var in the next Examples is returned as matrix object var;
//Reading whole variable, if the variable is scalar:
//Parameters: "BAUD" - controller variable name
//(optional:)ProgramBuffer.ACSC_NONE - ACSPL+ variable
var = Ch.ReadVariableAsMatrix("BAUD");
//or:
var = Ch.ReadVariableAsMatrix("BAUD",
    ProgramBuffer.ACSC_NONE);
//now var is a matrix of size 1x1
//Reading whole variable, if the variable is one-dimensional
//array:
//Parameters: "VEL" - controller variable name
//(optional:)ProgramBuffer.ACSC_NONE - ACSPL+ variable
var = Ch.ReadVariableAsMatrix("VEL");
//or:
var = Ch.ReadVariableAsMatrix("VEL",
    ProgramBuffer.ACSC_NONE);
//now var is a matrix of size 1x8
//Reading whole variable, if the variable is two-dimensional
//array:
//Assumed that MyMatrix variable is declared as global
//MyMatrix(3)(4) in the controller:
//Parameters: "MyMatrix" – user variable name
//(optional:)ProgramBuffer.ACSC_NONE - global variable
var = Ch.ReadVariableAsMatrix("MyMatrix");
//or:
var = Ch.ReadVariableAsMatrix("MyMatrix",
    ProgramBuffer.ACSC_NONE);
//now var is a matrix of size 3x4
//Reading sub-vector (more then one element) from vector:
//Reading sub-vector with indexes from 1 to 5:
//Parameters: "VEL" - controller variable name
//ProgramBuffer.ACSC_NONE - ACSPL+ variable
//fromIndex – from11
//toIndex – to1
int fromIndex = 1;
int toIndex = 5;
var = Ch.ReadVariableAsMatrix("VEL",
    ProgramBuffer.ACSC_NONE, fromIndex, toIndex);
//now var is a matrix of size 1x5
```

```
//Reading sub-matrix from matrix:
//Reading sub-matrix from with rows from 0 to 1 and colomns
//from 0 to 2
//Assumed that MyMatrix variable is declared as global
//MyMatrix(3)(4) in the controller:
//Parameters: "MyMatrix” – user variable name
//ProgramBuffer.ACSC_NONE - global variable
//fromRowIndex – from1
//toRowIndex – to1
//fromColumnIndex – from2
//toColumnIndex - to2
int fromRowIndex = 0;
int toRowIndex = 1;
int fromColumnIndex = 0;
int toColumnIndex = 2;
var = Ch.ReadVariableAsMatrix("MyMatrix",
    ProgramBuffer.ACSC_NONE, fromRowIndex,
    toRowIndex, fromColumnIndex, toColumnIndex);
//Now var is a matrix 2x3
```

3.5.5 WriteVariable

Description

The method writes a value defined as object to the controller variable(s).

Syntax

object.WriteVariable(object value, string variable, [ProgramBuffer nBuf], [int from1],[int to1], [int from2], [int to2])

Async Syntax

ACSC_WAITBLOCK object.WriteVariableAsync(object value, string variable, [ProgramBuffer nBuf], [int from1],[int to1], [int from2], [int to2])

Arguments

value	Value to be assigned to the **variable**. The value can contain an integer or real number(s). The **value** can be scalar, one-dimensional array (vector) or two-dimensional array (matrix).
variable	Name of the controller variable
nBuf	Number of program buffer for local variable or **ProgramBuffer.ACSC_NONE** for global and ACSPL+ variables.
from1, to1	Index range (first dimension) starting from zero of **variable** (not of **value**). Default value: **Api.ACSC_NONE**.
from2, to2	Index range (second dimension) starting from zero of **variable** (not of value). Default value: **Api.ACSC_NONE**.

Return Value

None

Remarks

This method writes **value** to the specified **variable**. **variable** can be scalar, vector (one- dimensional array) or matrix (two-dimensional array).

variable can be either an ASCPL+ variable, user global variable or a user local variable. ASCPL+ and user global variables have global scope; therefore **nBuf** must be must be omitted or specified as **ProgramBuffer.ACSC_NONE** (–1) for these classes of variables.

User local variable exist only within a buffer. The buffer number must be specified for user local variable.

The index parameters can be specified in many different ways. Write the entire variable,a single element of the variable or several elements of the variable as follows:

- Write value to whole variable

  from1, **to1**, **from2** and **to2** indexes should be omitted or specified as **Api.ACSC_NONE**. The **value** dimension should correspond to the **variable** dimension as follows:

  - If **variable** is scalar, **value** should be scalar, or vector of size 1, or matrix of size 1x1
  - If **variable** is vector of size N, **value** should be vector of size N, or matrix of size 1xN, or matrix of size Nx1
  - If **variable** is matrix of size NxM, **value** should be matrix of size NxM
- Writing a value to one element of vector (one-dimensional array)

  The **from1** and **to1** should be equal and specify index of the element. **from2** and **to2** should be omitted or specified as **Api.ACSC_NONE**.
- Writing value to one element of matrix (two-dimensional array)

  The **from1** and **to1** should be equal and specify row number. The **from2** and **to2** should be equal and specify column number.
- Writing a value to sub-vector (more then one element) of vector

  The **from1** and **to1** should specify sub-vector index range, from1 should be less then **to1.** The **from2** and **to2** should be omitted or specified as **Api.ACSC_NONE**.
- Writing a value to row or part of row of matrix

  The **from1** and **to1** should be equal and specify row number. The **from2** and **to2** should specify elements range within the specified row, **from2** should be less then **to2**.
- Writing a value to column or part of column of matrix

  The **from1** and **to1** should specify rows range within the specified column, **from1** should be less than **to1**. The **from2** and **to2** should be equal and specify column number.
- Writing a value to sub-matrix of matrix

  The **from1** and **to1** should specify rows range, **from1** should be less then **to1**. The **from2** and **to2** should specify columns range, **from2** should be less then **to2**.

If indexes (**from1**, **to1**, **from2**, **to2**) are specified, their values must correspond to value dimensions described below:

Table 3-10. Value Dimension Indices

Dimension	Indices
Scalar	from1=to1 and from2=to2
Vector of size N	(to1-from1+1=N and to2=from2) or (to1=from1 and to2-from2+1=N)
Matrix of size NxM	to1-from1+1=N and to2-from2+1=M
Scalar	from1=to1 and from2=to2
Vector of size N	(to1-from1+1=N and to2=from2) or (to1=from1 and to2-from2+1=N)

value type is not required to be the same as **variable** type. The library provides automatic conversion from integer to real and from real to integer.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Write variable
//Initialize scalar
int scalar = 57600;
//Initialize vector
double[] vector = new double[8];
for (int i = 0; i < vector.Length; i++)
{
    vector[i] = 1.1;
}
//Initialize matrix
int[,] matrix = new int[3, 4];
for (int i = 0; i <= matrix.GetUpperBound(0); i++)
{
    for (int j = 0; j <= matrix.GetUpperBound(1); j++)
    {
        matrix[i, j] = 2;
    }
}
//Initialize subVector
int[] subVector = new int[] { 3, 3 };
//Initialize subMatrix
double[,] subMatrix = new double[,] { { 4.0, 4.0 },
    { 4.0, 4.0 } };
//Writing scalar to whole variable:
//Parameters: scalar - value to be assigned to the variable
//"BAUD" - controller variable name
//(optional:)ProgramBuffer.ACSC_NONE - ACSPL+ variable
```

```
Ch.WriteVariable(scalar, "BAUD");
//or:
Ch.WriteVariable(scalar, "BAUD", ProgramBuffer.ACSC_NONE);
//Writing vector (one-dimensional array) to the whole
//variable:
//Parameters: vector - value to be assigned to the variable
//"VEL" - controller variable name
//(optional:)ProgramBuffer.ACSC_NONE - ACSPL+ variable
Ch.WriteVariable(vector, "VEL");
//or:
Ch.WriteVariable(vector, "VEL", ProgramBuffer.ACSC_NONE);
//Writing matrix to whole variable:
//Assumed that MyMatrix variable is declared as global
//MyMatrix(3)(4) in the controller:
//Parameters: matrix - value to be assigned to the variable
//"MyMatrix" - user variable name
//(optional:)ProgramBuffer.ACSC_NONE - global variable
Ch.WriteVariable(matrix, "MyMatrix");
//or:
Ch.WriteVariable(matrix, "MyMatrix",
    ProgramBuffer.ACSC_NONE);
//Writing value to one element of vector (one-dimensional
//array):
//Parameters: scalar - value to be assigned to the variable
//"VEL" - controller variable name
//ProgramBuffer.ACSC_NONE - ACSPL+ variable
//elementIndex - from1, to1
int elementIndex = 3;
Ch.WriteVariable(scalar, "VEL", ProgramBuffer.ACSC_NONE,
    elementIndex, elementIndex);
//Writing value to one element of matrix (two-dimensional
//array):
//Writing value to one element of matrix from row 1,
//column 2:
//Assumed that MyMatrix variable is declared as global
//MyMatrix(3)(4) in the controller:
//Parameters: scalar - value to be assigned to the variable
//"MyMatrix" - user variable name
//ProgramBuffer.ACSC_NONE - global variable
//elementRow - from1, to1
//elementColumn – from2, to2
int elementRow = 1;
int elementColumn = 2;
Ch.WriteVariable(scalar, "MyMatrix",
    ProgramBuffer.ACSC_NONE, elementRow, elementRow,
    elementColumn, elementColumn);
//Writing value to sub-vector (more then one element) of
//vector:
//Writing sub-vector with indexes from 2 to 3:
```

```
//Parameters: subVector - value to be assigned to the
//variable
//"VEL" - controller variable name
//ProgramBuffer.ACSC_NONE - ACSPL+ variable
//fromIndex - from1
//toIndex - to1
int fromIndex = 2;
int toIndex = 3;
Ch.WriteVariable(subVector, "VEL", ProgramBuffer.ACSC_NONE,
    fromIndex, toIndex);
//Writing value to row or part of row of matrix:
//Writing value to row 1, from element with index 0 till
//element with index 1:
//Assumed that MyMatrix variable is declared as global
//MyMatrix(3)(4) in the controller:
//Parameters: subVector - value to be assigned to the
//variable
//"MyMatrix" - user variable name
//ProgramBuffer.ACSC_NONE - global variable
//rowIndex - from1, to1
//fromColumnIndex - from2
//toColumnIndex - to2
int rowIndex = 1;
int fromColumnIndex = 0;
int toColumnIndex = 1;
Ch.WriteVariable(subVector, "MyMatrix",
    ProgramBuffer.ACSC_NONE, rowIndex, rowIndex,
    fromColumnIndex, toColumnIndex);
//Writing value to column or part of column of matrix:
//Writing value to column 2, from element with index 0 till
//element with index 1:
//Assumed that MyMatrix variable is declared as global
//MyMatrix(3)(4) in the controller:
//Parameters: subVector - value to be assigned to the
//variable
//"MyMatrix" - user variable name
//ProgramBuffer.ACSC_NONE - global variable
//fromRowIndex - from1
//toRowIndex - to1
//columnIndex - from2, to2
int fromRowIndex = 0;
int toRowIndex = 1;
int columnIndex = 2;
Ch.WriteVariable(subVector, "MyMatrix",
    ProgramBuffer.ACSC_NONE, fromRowIndex,
    toRowIndex, columnIndex, columnIndex);
//Writing value to sub-matrix of matrix:
//Writing sub-matrix to rows from 2 to 3 and colomns from 1
//to 2:
```

```
//Assumed that MyMatrix variable is declared as global
//MyMatrix(3)(4) in the controller:
//Parameters: subMatrix - value to be assigned to the
//variable
//"MyMatrix" - user variable name
//ProgramBuffer.ACSC_NONE - global variable
//fromRowIndex - from1
//toRowIndex – to1
//fromColumnIndex – from2
//toColumnIndex – to2
fromRowIndex = 1;
toRowIndex = 2;
fromColumnIndex = 2;
toColumnIndex = 3;
Ch.WriteVariable(subMatrix, "MyMatrix",
    ProgramBuffer.ACSC_NONE, fromRowIndex, toRowIndex,
    fromColumnIndex, toColumnIndex);
```

3.5.6 ReadString

Description

The function retrieves String type values from ACSPL standard / user defined String variables/arrays.

Syntax

Object.ReadString(ProgramBuffer nBuf, string var, int from1, int to1, int from2, int to2)

Async Syntax

ACSC_WAITBLOCK Object.ReadStringAsync(ProgramBuffer nBuf, string var, int from1, int to1, int from2, int to2)

Arguments

NBuf	Number of program buffer for local variable or ACSC_NONE (-1) for global and standard variable.
Var	Pointer to a character string that contains a name of the variable.
From1	Index range (first dimension).
To1	Index range (first dimension).
From2	Index range (second dimension).
To2	Index range (second dimension).

Return Value

Object

Example

```
object obj = _API.ReadString(ProgramBuffer.ACSC_NONE, "str1", -1, -1, -1,
-1);
```

3.5.7 WriteString

The function writes values to ACSPL standard or user-defined String variables or String arrays.

Syntax

Object.WriteString(object obj, ProgramBuffer nBuf, string var, int from1, int to1, int from2, int to2)

Async Syntax

ACSC_WAITBLOCK Object.WriteStringAsync(object obj, ProgramBuffer nBuf, string var, int from1, int to1, int from2, int to2)

Arguments

obj	Object, source of string data to be written
nBuf	Number of program buffer for local variable or ACSC_NONE (-1) for global and standard variable.
var	Pointer to a character string that contains a name of the variable.
From1	Index range (first dimension).
To1	Index range (first dimension).
From2	Index range (second dimension).
To2	Index range (second dimension).

Return Value

None

Example

```
Api acsApi = new Api();
acsApi.OpenCommEthernetTCP("10.0.0.108", 701);
ACSC_WAITBLOCK wait; object retObj;
List<string< ActualStrings = new List<string>();
string expectedString = "LIB TEST";
string expectedStringArray = ".NET LIB Test";
string stringVariable = "String myStringVariable(15)";
string stringArray = "GLOBAL String myStringArray(16)(2)";
bool flag = false;
bool is_Async = false;
acsApi.ClearBuffer(ProgramBuffer.ACSC_BUFFER_0, 0, 1000);
Thread.Sleep(500);
acsApi.AppendBuffer(ProgramBuffer.ACSC_BUFFER_0, stringVariable);
```

```
Thread.Sleep(500);
acsApi.CompileBuffer(ProgramBuffer.ACSC_BUFFER_0);
Thread.Sleep(500);
if (is_Async.Equals(true))
{
   wait = acsApi.WriteStringAsync("LIB TEST", ProgramBuffer.ACSC_BUFFER_
0, "myStringVariable");
   retObj = acsApi.GetResult(wait, 5000);
   wait = acsApi.ReadStringAsync(ProgramBuffer.ACSC_BUFFER_0,
"myStringVariable");
   retObj = acsApi.GetResult(wait, 5000);
   string[] temp = retObj as string[];
   if (temp.Length > 0)
   {
      ActualStrings.Add(temp[0]);
   }
   acsApi.AppendBuffer(ProgramBuffer.ACSC_BUFFER_0, stringArray);
   Thread.Sleep(500);
   acsApi.CompileBuffer(ProgramBuffer.ACSC_BUFFER_0);
   Thread.Sleep(500);
   wait = acsApi.WriteStringAsync(".NET LIB Test", ProgramBuffer.ACSC_
BUFFER_0, "myStringArray", 0, 0);
   retObj = acsApi.GetResult(wait, 5000); wait = acsApi.ReadStringAsync
(ProgramBuffer.ACSC_BUFFER_0, "myStringArray", 0, 0);
   retObj = acsApi.GetResult(wait, 5000); temp = retObj as string[];
   if (temp.Length > 0)
   {
      ActualStrings.Add(temp[0]);
   }
}
else
{
   acsApi.WriteString("LIB TEST", ProgramBuffer.ACSC_BUFFER_0,
"myStringVariable");
   StringBuilder[] retString = (StringBuilder[])acsApi.ReadString
(ProgramBuffer.ACSC_BUFFER_0, "myStringVariable");
   ActualStrings.Add(retString[0].ToString()); acsApi.AppendBuffer
(ProgramBuffer.ACSC_BUFFER_0, stringArray);
   Thread.Sleep(500);
   acsApi.CompileBuffer(ProgramBuffer.ACSC_BUFFER_0);
   Thread.Sleep(500);
   acsApi.WriteString(".NET LIB Test", ProgramBuffer.ACSC_BUFFER_0,
"myStringArray", 0, 0);
   retString = (StringBuilder[])acsApi.ReadString(ProgramBuffer.ACSC_
BUFFER_0, "myStringArray", 0, 0); ActualStrings.Add(retString[0].ToString
());
}
if (ActualStrings[0].Equals(expectedString) && ActualStrings[1].Equals
(expectedStringArray)) { flag = true; }
```

3.5.8 ReadStruct

Description

The function retrieves struct values from ACSPL+ struct variables or arrays.

Syntax

Object.ReadStruct([ProgramBuffer nBuf], string var, int from, int to, Type myStruct)

Object.ReadStruct([ProgramBuffer nBuf], string var, int from1, int to1, int from2, int to2, Type myStruct)

Async Syntax

ACSC_WAITBLOCK Object.ReadStruct([ProgramBuffer nBuf], string var, int from, int to, Type myStruct)

ACSC_WAITBLOCK Object.ReadStruct([ProgramBuffer nBuf], string var, int from1, int to1, int from2, int to2, Type myStruct)

Arguments

nBuf	Number of program buffer for local struct variable or ACSC_NONE (-1) for global struct variable.
var	String that contains the name of the ACSPL struct variable.
from, to	Index range.
from1, to1	Index range of dimension 1.
from2, to2	Index range of dimension 2.
myStruct	Type of user struct.

Return Value

Object

Comments

Extended error information can be obtained by calling **GetLastError**.

Example

```
//struct should be separate from the main function
[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct myGlobalStruct
{
   public int var1;
   public double var2;
   [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 7)]
   // The string should be marshaled as a fixed-length character array.
   // SizeConst = 7: This parameter specifies the size of the character
array that
   // will be used to marshal the string (depends on ACSPL string
```

```
variable size).
   public string var3;

object obj;
myGlobalStruct myStruct = new myGlobalStruct();
// globalStruct is a struct variable defined in DBuffer
obj = api.ReadStruct(ProgramBuffer.ACSC_NONE, "globalStruct", 0, 0,
typeof(myGlobalStruct));
// Asynchronous ReadStruct
ACSC_WAITBLOCK wait = api.ReadStructAsync(ProgramBuffer.ACSC_NONE,
"globalStruct", 0, 0, typeof(myGlobalStruct));
obj = api.GetResult(wait, 5000, typeof(myGlobalStruct));
```

3.5.9 WriteStruct

Description

The function writes struct values to ACSPL+ struct variables or arrays.

Syntax

Object.WriteStruct([ProgramBuffer nBuf], string var, int from, int to,T myStruct)

Object.WriteStruct([ProgramBuffer nBuf], string var, int from1, int to1, int from2, int to2, T myStruct)

Async Syntax

ACSC_WAITBLOCK Object.WriteStruct([ProgramBuffer nBuf], string var, int from, int to, T myStruct)

ACSC_WAITBLOCK Object.WriteStruct([ProgramBuffer nBuf], string var, int from1, int to1, int from2, int to2, T myStruct)

Arguments

nBuf	Number of program buffer for local struct variable or ACSC_NONE (-1) for global struct variable.
var	String that contains the name of the ACSPL struct variable.
from, to	Index range.
from1, to1	Index range of dimension 1.
from2, to2	Index range of dimension 2.
myStruct	Struct that contains the data to be written ACSPL struct variable.

Return Value

None

Example

```
//struct should be separate from the main function
[StructLayout(LayoutKind.Sequential, Pack = 1)]
```

```
public struct myGlobalStruct
{
   public int var1;
   public double var2;
   [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 6)]
   // The string should be marshaled as a fixed-length character array.
   // SizeConst = 7: This parameter specifies the size of the character
array that
   // will be used to marshal the string (depends on ACSPL string
variable size).
   public string var3;

myGlobalStruct myStruct = new myGlobalStruct();
myStruct.var1 = 111;
myStruct.var2 = 22.2;
myStruct.var3 = "hello";
//globalStruct is struct variable that is defined in DBuffer
api.WriteStruct(ProgramBuffer.ACSC_NONE, "globalStruct", 0, 0,
myStruct);
```

3.6 History Buffer Management Methods

The History Buffer Management methods are:

Table 3-11. History Buffer Management Methods

Methods	Description
OpenHistoryBuffer	Opens a history buffer.
CloseHistoryBuffer	Closes a history buffer.
GetHistory	Retrieves the contents of the history buffer.

3.6.1 OpenHistoryBuffer

Description

The method opens a history buffer.

Syntax

object.OpenHistoryBuffer(int Size)

Arguments

size	Required size of the buffer in bytes

Return Value

None

Remarks

The method allocates a history buffer that stores all commands sent to the controller and all responses and unsolicited messages received from the controller.

Only one history buffer can be open for each communication channel.

The buffer works as a cyclic buffer. When the amount of the stored data exceeds the buffer size, the newly stored data overwrites the earliest data in the buffer.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Open history buffer
int bufferSize = 5000;
//The method opens an history buffer
//size of the buffer is 5000
Ch.OpenHistoryBuffer(bufferSize);
```

3.6.2 CloseHistoryBuffer

Description

The method closes the history buffer and discards all stored history.

Syntax

object.CloseHistoryBuffer()

Arguments

None

Return Value

None

Remarks

The method closes the history buffer and releases the used memory. All information stored in the buffer is discarded.

If the method fails, the Error object is filled with the Error Description.

Example

```
            // Close history buffer
            //The method closes an history buffer
            Ch.CloseHistoryBuffer();
```

3.6.3 GetHistory

Description

The method retrieves the contents of the history buffer.

Syntax

object.GetHistory(bool bClear)

Arguments

bClear	If TRUE, the method clears contents of the history buffer. If FALSE, the history buffer content is not cleared.

Return Value

String

The method retrieves the communication history from the history buffer.

Remarks

The communication history includes all commands sent to the controller and all responses and unsolicited messages from the controller. The amount of history data is limited by the size of the history buffer. The history buffer works as a cyclic buffer: when the amount of the stored data exceeds the buffer size, the newly stored data overwrites the earliest data in the buffer.

Therefore, as a rule, the retrieved communication history includes only recent commands, responses and unsolicited messages. The depth of the retrieved history depends on the history buffer size.

The history data is retrieved in historical order, i.e. the earliest message is stored at the beginning of the returned string. The beginning of the return string might be incomplete, if it has been partially overwritten in the history buffer.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Get history buffer
int bufferSize = 5000;
string cmd = "enable 0";
//The method opens an history buffer
//size of the buffer is 5000
Ch.OpenHistoryBuffer(bufferSize);
//Sending the GUI "enable 0"
Ch.Send(cmd);
//The method retrieves the contents of the
//history buffer.
string str = Ch.GetHistory(false);
//The method closes the history buffer
Ch.CloseHistoryBuffer();
```

3.7 Unsolicited Messages Buffer Management Methods

The Unsolicited Messages Buffer Management methods are:

Table 3-12. Unsolicited Messages Buffer Management Methods

Methods	Description
OpenMessageBuffer	Opens an unsolicited messages buffer.
CloseMessageBuffer	Closes an unsolicited messages buffer.
GetSingleMessage	Retrieves single unsolicited message from the buffer.

3.7.1 OpenMessageBuffer

Description

The method opens an unsolicited messages buffer.

Syntax

object.OpenMessageBuffer(int size)

Arguments

size	Required size of the buffer in bytes

Return Value

None

Remarks

The method allocates a buffer that stores unsolicited messages from the controller.

Unsolicited messages are messages that the controller sends on its own initiative and not as a response to a command. For Example, the **disp** command in an ACSPL+ program forces the controller to send an unsolicited message.

Only one message buffer can be open for each communicationchannel.

The message buffer works as a FIFO buffer: **GetSingleMessage** extracts the earliest message stored in the buffer. If **GetSingleMessage** extracts the messages slower than the controller produces them, buffer overflow can occur, and some messages will be lost. Generally, the greater the buffer, the less likely is buffer overflow to occur.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Open message buffer
int bufferSize = 5000;
//The method opens an unsolicited messages
//buffer size of the opened buffer is 5000
Ch.OpenMessageBuffer(bufferSize);
```

3.7.2 CloseMessageBuffer

Description

The method closes the messages buffer and discards all stored unsolicited messages.

Syntax

object.CloseMessageBuffer()

Arguments

None

Return Value

None

Remarks

The method closes the message buffer and releases the used memory. All unsolicited messages stored in the buffer are discarded.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Close message buffer
//The method closes the messages buffer
//and discards all stored unsolicited messages
Ch.CloseMessageBuffer();
```

3.7.3 GetSingleMessage

Description

The method retrieves single unsolicited message from the buffer. This method only works if you setup a buffer using *OpenMessageBuffer*. If there is no message in the buffer the method waits until the message arrives or time out expires.

Syntax

object.GetSingleMessage(int timeout)

Arguments

timeout	Maximum waiting time in milliseconds.

Return Value

String

Remarks

The method retrieves single unsolicited message from the message buffer.

If no, unsolicited message is received the method waits until a message arrives. If the timeout expires, the method exits with **ACSC_TIMEOUT** error.

If the method fails, the Error object is filled with the Error Description.

Example

```
            int bufferSize = 5000;
            int timeout = 1000;
            //The method opens an unsolicited messages
```

```
//buffer, size of the opened buffer is 5000
Ch.OpenMessageBuffer(bufferSize);

string program = "disp\"Test Message\"; stop\n";
// The method appends one ACSPL+ line to buffer 0
Ch.AppendBuffer(ProgramBuffer.ACSC_BUFFER_0, program);
// Get program of buffer 0
string buf = Ch.UploadBuffer(ProgramBuffer.ACSC_BUFFER_0);
Ch.CompileBuffer(ProgramBuffer.ACSC_BUFFER_0);
Ch.RunBuffer(ProgramBuffer.ACSC_BUFFER_0,null);

//The method retrieves unsolicited message
//from the buffer and wait 1 second till a
//message arrives
string message = Ch.GetSingleMessage(timeout);

//The method closes the messages buffer
//and discards all stored unsolicited messages
Ch.CloseMessageBuffer();

Ch.ClearBuffer(ProgramBuffer.ACSC_BUFFER_0, 1, 1000);
```

3.8 Log File Management Methods

The Log File Management methods are:

Table 3-13. Log File Management Methods

Method	Description
OpenLogFile	Opens a log file.
CloseLogFile	Closes a log file.
WriteSCLogFile	Writes a log file.
FlushLogFile	Flushes log to file.

3.8.1 OpenLogFile

Description

The method opens a log file.

Syntax

object.OpenLogFile(string filename)

Arguments

filename	String containing the name or path and name of the log file.

Return Value

None

Remarks

The method opens a binary file that stores all communication history. Only one log file can be open for each communication channel.

If the log file has been open, the library writes all incoming and outgoing messages to the specified file. The messages are written to the file in binary format, i.e., exactly as they are received and sent, including all service bytes.

Unlike the history buffer, the log file cannot be read within the library. The main usage of the log file is for debug purposes.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Open log file
Ch.OpenLogFile("C:\\COMLogFile.log");
```

3.8.2 CloseLogFile

Description

The method closes the log file.

Syntax

object.CloseLogFile()

Arguments

None

Return Value

None

Remarks

An application must always call CloseLogFile before it exits. Otherwise, the data written to the file might be lost.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Close log file
//The method opens the log file "COMLogFile.log"
Ch.OpenLogFile("C:\\COMLogFile.log");
//The method closes the log file "COMLogFile.log"
Ch.CloseLogFile();
```

3.8.3 WriteLogFile

Description

The method writes to log file.

Syntax

object.WriteLogFile(string buf)

Arguments

buf	The string to be written to log file.

Return Value

None

Remarks

The method writes data from a buffer to log file. The log file should be previously opened by the *OpenLogFile* method.

The method adds its Arguments to the internal UMD binary buffer. In previous C Library versions, the log file had to be explicitly opened by the *OpenLogFile* method, otherwise the function would return an error. Starting with new versions, this is no longer the case. See FlushLogFile.

If the method fails, the Error object is populated with the Error Description.

Example

```
// Open log file
//Check you have a permision write a file to this place
Ch.OpenLogFile("C:\\Test\\LogFile.log");
//Write to log file a text "Writing to log file";
Ch.WriteLogFile("Writing to log file");
//Close the log file
Ch.CloseLogFile();
```

3.8.4 FlushLogFile

Description

The method flushes log to file.

Syntax

object.FlushLogFile(string fileName)

Arguments

fileName	String specifying the text file into which the buffer is to be flushed.

Return Value

None

Remarks

If Continuous Log is active, the function will fail.

Example

```
// Open log file
//Check you have a permision write a file to this place
Ch.OpenLogFile("C:\\Test\\LogFile.log");
//Write to log file a text "First writing to log file";
Ch.WriteLogFile("First writing to log file");
//Write to log file a text "Second writing to log file";
Ch.WriteLogFile("Second writing to log file");
//Close the log file
Ch.CloseLogFile();
// Flush log file
Ch.FlushLogFile("C:\\Test\\NewLogFile.log");
```

3.9 SPiiPlusSC Log File Management Methods

These methods can only be used with the SPiiPlusSC Motion Controller.

The SPiiPlusSC Log File Management methods are:

Table 3-14. SPiiPlusSC Log File Management Methods

Method	Description
OpenSCLogFile	Used for opening the SPiiPlusSC log file.
CloseSCLogFile	Used for closing the SPiiPlusSC log file.
WriteSCLogFile	Used for writing to the SPiiPlusSC log file.
FlushSCLogFile	Enables flushing the SPiiPlusSC internal buffer to a specified text file from the NET Library application.

3.9.1 OpenSCLogFile

Description

The method opens the SPiiPlusSC log file.

Syntax

object.OpenSCLogFile(string filename);

Arguments

filename	The name or path of the log file.

Return Value

None

Remarks

The method opens a binary file that stores all SPiiPlusSC log history. The messages are written to the file in binary format, i.e., exactly as they are received and sent, including all service bytes.

The main use of the SPiiPlusSC log file is for debugging purposes.

Example

```
// Open SPiiPlusSC log file
Ch.OpenSCLogFile("C:\\SCLogFile2.txt");
```

3.9.2 CloseSCLogFile

Description

The method closes the SPiiPlusSC log file.

Syntax

object.CloseSCLogFile();

Arguments

None

Return Value

None

Remarks

An application must always call **CloseSCLogFile** before it exits; otherwise, the data written to the file might be lost.

Example

```
// Close SPiiPlusSC log file
Ch.CloseSCLogFile();
```

3.9.3 WriteSCLogFile

Description

The method writes to the SPiiPlusSC log file.

Syntax

object.WriteSCLogFile(string buf);

Arguments

buf	The string to be written to the log file.

Return Value

None

Remarks

The method writes data from a buffer to the SPiiPlusSC log file.

Example

```
// Write to SPiiPlusSC log file
Ch.WriteSCLogFile("Hello World!");
```

3.9.4 FlushSCLogFile

Description

The method enables flushing the SPiiPlusSC internal buffer to a specified text file from the .NET Library application.

Syntax

object.FlushSCLogFile(string filename, bool bClear);

Arguments

filename	String that specifies the file name.
bClear	Can be TRUE or FALSE: TRUE – the method clears contents of the log buffer. FALSE – the log buffer content is not cleared.

Return Value

None

Remarks

If Continuous Log is active, the function will fail.

Example

```
// Flush to SPiiPlusSC log file
Ch.FlushSCLogFile("C:\\SCLogFile3.txt", true);
```

3.10 Shared Memory Methods

The Shared Memory methods have been added in support of SPiiPlusSC Motion Controller to enable accessing shared memory addresses. They cannot be used with any other SPiiPlus family product.

The Shared Memory methods are:

Table 3-15. Shared Memory Methods

Method	Description
GetSharedMemoryAddress	Reads the address of shared memory variable.
ReadSharedMemoryInteger	Reads value(s) from an integer shared memory variable.

Method	Description
ReadSharedMemoryReal	Reads value(s) from a real shared memory variable.
WriteSharedMemoryVariable	Writes value(s) to a real or integer shared memory variable.

3.10.1 GetSharedMemoryAddress

Description

The method reads the address of shared memory variable.

Syntax

object.GetSharedMemoryAddress(ProgramBuffer nBuf, string var);

Async Syntax

ACSC_WAITBLOCK object.GetSharedMemoryAddressAsync(ProgramBuffer nBuf, string var);

Arguments

nBuf	Number of program buffer for local variable or ACSC_NONE for global and ACSPL+ variable.
var	Pointer to a buffer that contains a name of the variable.

Return Value

Uint.

Example

See Shared Memory Program Example.

3.10.2 ReadSharedMemoryInteger

Description

The method reads value(s) from an integer shared memory variable.

Syntax

object.ReadSharedMemoryInteger(uint address, int from1, int to1, int from2, int to2, int[] values);

Arguments

address	Shared memory address of the variable that should be read.
from1, to1	Index range (first dimension) for one dimensional array variables.
from2, to2	Index range (second dimension) for two dimensional matrix

Return Value

object.

Returned value is a scalar or an array of integers

Example

See Shared Memory Program Example.

3.10.3 ReadSharedMemoryReal

Description

The method reads value(s) from a real shared memory variable.

Syntax

object.ReadSharedMemoryReal(uint address, int from1, int to1, int from2, int to2, double[] values);

Arguments

Address	Shared memory address of the variable that should be read.
from1, to1	Index range (first dimension) for one dimensional array variables.
from2, to2	Index range (second dimension) for two dimensional matrix

Return Value

object.

Returned value is a scalar or an array of doubles

Example

See Shared Memory Program Example.

3.10.4 WriteSharedMemoryVariable

Description

The method writes value(s) to the integer or real shared memory variable.

Syntax

object.WriteSharedMemoryVariable(object value, uint address, int from1, int to1, int from2, int to2);

Arguments

value	A value to be written. The value could be of integer type or real type or an array of integers or reals.
address	Shared memory address of the variable is to written to.
from1, to1	Index range (first dimension) for one dimensional array variables.
from2, to2	Index range (second dimension) for two dimensional matrix variables.
values	Buffer containing values that should be written.

Return Value

None

Example

See Shared Memory Program Example.

3.10.5 Shared Memory Program Example

```
int[] intValues = new int[] { 1, 2, 3, 4 };
int[] intResults = new int[4];

uint address1 = Ch.GetSharedMemoryAddress(ProgramBuffer.ACSC_NONE,
"intVariable");

Ch.WriteSharedMemoryVariable(intValues, address1, 0, 1, 0, 1);
intResults = (int[])Ch.ReadSharedMemoryInteger(address1, 0, 1, 0, 1);

double[] realValues = new double[] { 666.66, 777.77, 888.88, 999.99 };
double[] realResults = new double[4];

uint address2 = Ch.GetSharedMemoryAddress(ProgramBuffer.ACSC_NONE,
"realVariable");

Ch.WriteSharedMemoryVariable(realValues, address2, 0, 1, 0, 1);
realResults = (double[])Ch.ReadSharedMemoryReal(address2, 0, 1, 0, 1);
```

3.11 System Configuration Methods

The System Configuration methods are:

Table 3-16. System Configuration Methods

Method	Description
SetConf	The method writes system configuration data.
GetConf	The method reads system configuration data.
GetVolatileMemoryUsage	The function retrieves the volatile memory load in percentage.
GetVolatileMemoryTotal	The function retrieves the amount of total volatile memory.
GetVolatileMemoryFree	The function retrieves the amount of free volatile memory.
GetNonVolatileMemoryUsage	The function retrieves the non-volatile memory load in percentage.
GetNonVolatileMemoryTotal	The function retrieves the amount of total non-volatile memory.

Method	Description
GetNonVolatileMemoryFree	The function retrieves the amount of free non-volatile memory.

3.11.1 SetConf

Description

The method writes system configuration data.

Syntax

object.SetConf(int key, int index, double value)

Async Syntax

ACSC_WAITBLOCK ACSC_WAITBLOCK object.SetConfAsync(int key, int index, double value)

Arguments

key	**Configuration Key** (see Axis Definitions that specifies the configured feature. Assigns value of key argument in ACSPL+ **SetConf** command (see *SPiiPlus Command& Variable Reference Guide*).
index	Specifies corresponding axis or buffer number. Assigns value of index
value	Value to write to specified key. Assigns value of value argument in ACSPL+ **SetConf** command.

Return Value

None

Remarks

The method writes system configuration data. **key** specifies the feature number and **index** defines the axis or buffer to which it should be applied. Use ACSC_CONF_XXX properties in the value field. For detailed description of system configuration see the description of the **SetConf** method in the *SPiiPlus Command & Variable Reference Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous Set configuration of axis 0
//Writes system configuration data of axis 0
//to 1 with key 204
Ch.SetConf((int)ConfigKey.ACSC_CONF_MFLAGS9_KEY,
    (int)Axis.ACSC_AXIS_0, 1);
// Asynchronous Set configuration of axis 0
ACSC_WAITBLOCK wb =
```

```
Ch.SetConfAsync((int)ConfigKey.ACSC_CONF_MFLAGS9_KEY,
(int)Axis.ACSC_AXIS_0, 1);
```

3.11.2 GetConf

Description

The method reads system configuration data.

Syntax

object.GetConf(int key, int index)

Async Syntax

ACSC_WAITBLOCK object.GetConfAsync(int key, int index)

Arguments

key	**Configuration Key** (see Axis Definitions) that specifies the configured feature. Assigns value of key argument in ACSPL+ **GetConf** command (see *SPiiPlus Command & Variable Reference Guide*).
index	Specifies corresponding axis or buffer number. Assigns value of index

Return Value

Double

The method reads system configuration data.

Remarks

key specifies the feature number and index defines axis or buffer to which it should be applied. For detailed *Description* of system configuration see *SPiiPlus Command & Variable Reference Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous Get configuration of axis 0
//Reads system configuration data of axis 0
//with key 204
double configuration = api.GetConf(
    (int)ConfigKey.ACSC_CONF_MFLAGS9_KEY,
    (int)Axis.ACSC_AXIS_0);
// Asynchronous Get configuration of axis 0
int timeout = 2000;
ACSC_WAITBLOCK wb = api.GetConfAsync(
    (int)ConfigKey.ACSC_CONF_MFLAGS9_KEY,
    (int)Axis.ACSC_AXIS_0);
configuration = (double)api.GetResult(wb, timeout);
```

3.11.3 GetVolatileMemoryUsage

Description

The function retrieves the volatile memory load in percentage.

Syntax

object.GetVolatileMemoryUsage()

Async Syntax

ACSC_WAITBLOCK object.GetVolatileMemoryUsageAsync()

Arguments

None

Return Value

Double

Example

```
// Synchronous Get volatile memory usage
double volatileMemoryUsage = api.GetVolatileMemoryUsage();

// Asynchronous Get volatile memory usage
int timeout = 2000;
ACSC_WAITBLOCK wb = api.GetVolatileMemoryUsageAsync();
double volatileMemoryUsage = (double)api.GetResult(wb, timeout);
```

3.11.4 GetVolatileMemoryTotal

Description

The function retrieves the amount of total volatile memory.

Syntax

object.GetVolatileMemoryTotal()

Async Syntax

ACSC_WAITBLOCK object.GetVolatileMemoryTotalAsync()

Arguments

None

Return Value

Double

Example

```
// Synchronous Get total volatile memory
double volatileMemoryTotal = Ch.GetVolatileMemoryTotal();

// Asynchronous Get total volatile memory
int timeout = 2000;
ACSC_WAITBLOCK wb = Ch.GetVolatileMemoryTotalAsync();
double volatileMemoryTotal = (double)Ch.GetResult(wb, timeout);
```

3.11.5 GetVolatileMemoryFree

Description

The function retrieves the amount of free volatile memory.

Syntax

object.GetVolatileMemoryFree()

Async Syntax

ACSC_WAITBLOCK object.GetVolatileMemoryFreeAsync()

Arguments

None

Return Value

Double

Example

```
// Synchronous Get free volatile memory
double volatileMemoryFree = Ch.GetVolatileMemoryFree();

// Asynchronous Get free volatile memory
int timeout = 2000;
ACSC_WAITBLOCK wb = Ch.GetVolatileMemoryFreeAsync();
double volatileMemoryFree = (double)Ch.GetResult(wb, timeout);
```

3.11.6 GetNonVolatileMemoryUsage

Description

The function retrieves the non-volatile memory load in percentage.

Syntax

object.GetNonVolatileMemoryUsage()

Async Syntax

ACSC_WAITBLOCK object.GetNonVolatileMemoryUsageAsync()

Arguments

None

Return Value

Double

Example

```
// Synchronous Get non volatile memory usage
double nonVolatileMemoryUsage = Ch.GetNonVolatileMemoryUsage();

// Asynchronous Get non volatile memory usage
int timeout = 2000;
ACSC_WAITBLOCK wb = Ch.GetNonVolatileMemoryUsageAsync();
double nonVolatileMemoryUsage = (double)Ch.GetResult(wb, timeout);
```

3.11.7 GetNonVolatileMemoryTotal

Description

The function retrieves the amount of total non-volatile memory.

Syntax

object.GetNonVolatileMemoryTotal()

Async Syntax

object.GetNonVolatileMemoryTotalAsync()

Arguments

None

Return Value

Double

Example

```
// Synchronous Get total non volatile memory
double nonVolatileMemoryTotal = Ch.GetNonVolatileMemoryTotal();

// Asynchronous Get total non volatile memory
int timeout = 2000;
ACSC_WAITBLOCK wb = Ch.GetNonVolatileMemoryTotalAsync();
double nonVolatileMemoryTotal = (double)Ch.GetResult(wb, timeout);
```

3.11.8 GetNonVolatileMemoryFree

Description

The function retrieves the amount of free non-volatile memory.

Syntax

object.GetNonVolatileMemoryFree()

Async Syntax

ACSC_WAITBLOCK object.GetNonVolatileMemoryFreeAsync()

Arguments

None

Return Value

Double

Example

```
// Synchronous Get free non volatile memory
double nonVolatileMemoryFree = Ch.GetNonVolatileMemoryFree();

// Asynchronous Get free non volatile memory
int timeout = 2000;
ACSC_WAITBLOCK wb = Ch.GetNonVolatileMemoryFreeAsync();
double nonVolatileMemoryFree = (double)Ch.GetResult(wb, timeout);
```

3.12 Setting and Reading Motion Parameters Methods

The Setting and Reading Motion Parameters methods are:

Table 3-17. Setting and Reading Motion Parameters Methods

Method	Description
SetVelocity	Defines a value of motion velocity.
SetVelocityImm	Defines a value of motion velocity having an immediate effect on any executed as well as impending motion.
GetVelocity	Retrieves a value of motion velocity.
SetAcceleration	Defines a value of motion acceleration.
SetAccelerationImm	Defines a value of motion acceleration having an immediate effect on any executed as well as impending motion.
GetAcceleration	Retrieves a value of motion acceleration.
SetDeceleration	Defines a value of motion deceleration.
SetDecelerationImm	Defines a value of motion deceleration having an immediate effect on any executed as well as impending motion.
GetDeceleration	Retrieves a value of motion deceleration.
SetJerk	Defines a value of motion jerk.
SetJerkImm	Defines a value of motion jerk having an immediate effect on any executed as well as impending motion.

Method	Description
GetJerk	Retrieves a value of motion jerk.
SetKillDeceleration	Defines a value of kill deceleration.
SetKillDecelerationImm	Defines a value of kill deceleration having an immediate effect on any executed as well as impending motion.
GetKillDeceleration	Retrieves a value of kill deceleration.
SetFPosition	Assigns a current value of feedback position.
GetFPosition	Retrieves a current value of motor feedback position.
SetRPosition	Assigns a current value of reference position.
GetRPosition	Retrieves a current value of reference position.
GetFVelocity	Retrieves a current value of motor feedback velocity.
GetRVelocity	Retrieves a current value of reference velocity.

3.12.1 SetVelocity

Description

The method defines a value of motion velocity.

Syntax

object.SetVelocity(Axis axis, double velocity)

Async Syntax

ACSC_WAITBLOCK object.SetVelocityImmAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
velocity	The value specifies required motion velocity. The value will be used in the subsequent motions except for the master-slave motions and the motions activated with the **ACSC_AMF_VELOCITY** flag.

Return Value

None

Remarks

The method writes the specified value to the controller. One value can be specified for each axis.

A single-axis motion uses the value of the corresponding axis. A multi-axis motion uses the value of the leading axis. The leading axis is an axis specified first in the motion command.

The method affects the motions initiated after the method call. The method has no effect on any motion that was started or planned before the method call. To change velocity of an executed or planned motion, use the *SetVelocityImm* method.

The method has no effect on the master-slave motions and the motions activated with the ACSC_AMF_VELOCITY flag.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous Set velocity of axis 0
double velocity = 10000;
//Sets the velocity to 10000 to axis 0
Ch.SetVelocity(Axis.ACSC_AXIS_0, velocity);

// Asynchronous Set velocity of axis 0
ACSC_WAITBLOCK wb = Ch.SetVelocityAsync(Axis.ACSC_AXIS_0, velocity);
```

3.12.2 SetVelocityImm

Description

The method defines a value of motion velocity. Unlike *SetVelocity*, the method has immediate effect on any executed or planned motion.

Syntax

object.SetVelocityImm(Axis axis, double velocity)

Async Syntax

ACSC_WAITBLOCK object.SetVelocityImmAsync(Axis axis, double velocity)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
velocity	The value specifies required motion velocity.

Return Value

None

Remarks

The method writes the specified value to the controller. One value can be specified for each axis.

A single-axis motion uses the value of the corresponding axis. A multi-axis motion uses the value of the leading axis. The leading axis is an axis specified first in the motion command.

The method affects:

> The currently executed motion. The controller provides a smooth transition from the instant current velocity to the specified new value.

- The waiting motions that were planned before the method call.
- The motions that will be commanded after the method call. The method has no effect on the master-slave motions.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous Set immediate velocity of axis 0
double velocity = 80000;
//Writes the specified value of velocity to the controller
Ch.SetVelocityImm(Axis.ACSC_AXIS_0, velocity);
// Asynchronous Set immediate velocity of axis 0
ACSC_WAITBLOCK wb Ch.SetVelocityImmAsync(Axis.ACSC_AXIS_0,velocity);
```

3.12.3 GetVelocity

Description

The method retrieves a value of motionvelocity.

Syntax

object.GetVelocity(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GetVelocityAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

Double

The method retrieves the value of the motion velocity.

Remarks

The retrieved value is the value defined by a previous call of *SetVelocity* or the default value if the method was not previously called.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous Get velocity of axis 0
// Retrieves a value of motion velocity of the 0 axis

double velocity = Ch.GetVelocity(Axis.ACSC_AXIS_0);

// Asynchronous Get velocity of axis 0

int timeout = 2000;
```

```
ACSC_WAITBLOCK wb = Ch.GetVelocityAsync(Axis.ACSC_AXIS_0);
double velocity = (double)Ch.GetResult(wb, timeout);
```

3.12.4 SetAcceleration

Description

The method defines a value of motion acceleration.

Syntax

object.SetAcceleration(Axis axis, double acceleration)

Async Syntax

ACSC_WAITBLOCK object. SetAccelerationImmAsync(Axis axis, doubleacceleration)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
acceleration	The value specifies required motion acceleration. The value will be used in the subsequent motions except the master-slave motions.

Return Value

None

Remarks

The method writes the specified value to the controller. One value can be specified for each axis.

A single-axis motion uses the value of the corresponding axis. A multi-axis motion uses the value of the leading axis. The leading axis is an axis specified first in the motion command.

The method affects the motions initiated after the method call. The method has no effect on any motion that was started or planned before the method call. To change acceleration of an executed or planned motion, use the *SetAccelerationImm* method.

The method has no effect on the master-slave motions.

If the method fails, the Error object is filled with the Error Description.

Example

```
 // Synchronous Set acceleration of axis 0
double acceleration = 200000;
//Defines a value of motion acceleration to 200000
Ch.SetAcceleration(Axis.ACSC_AXIS_0, acceleration);
// Asynchronous Set acceleration of axis 0
ACSC_WAITBLOCK wb = Ch.SetAccelerationAsync(Axis.ACSC_AXIS_0,
    acceleration);
```

3.12.5 SetAccelerationImm

Description

The method defines a value of motion acceleration. Unlike *SetAcceleration*, the method has immediate effect on any executed and planned motion.

Syntax

object.SetAccelerationImm(Axis axis, double acceleration)

Async Syntax

ACSC_WAITBLOCK object. SetAccelerationImmAsync(Axis axis, double acceleration)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions..
acceleration	The value specifies required motion acceleration.

Return Value

None

Remarks

The method writes the specified value to the controller. One value can be specified for each axis.

A single-axis motion uses the value of the corresponding axis. A multi-axis motion uses the value of the leading axis. The leading axis is an axis specified first in the motion command.

The method affects:

- The currently executed motion.
- The waiting motions that were planned before the method call.
- The motions that will be commanded after the method call. The method has no effect on the master-slave motions.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous Set immediate acceleration of axis 0
double acceleration = 300000;
//Writes the specified acceleration value to the controller
Ch.SetAccelerationImm(Axis.ACSC_AXIS_0, acceleration);
// Asynchronous Set immediate acceleration of axis 0
ACSC_WAITBLOCK wb = Ch.SetAccelerationImmAsync(
    Axis.ACSC_AXIS_0, acceleration);
```

3.12.6 GetAcceleration

Description

The method retrieves a value of motion acceleration.

Syntax

object.GetAcceleration(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GetAccelerationAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

Double

The method retrieves the value of the motion acceleration.

Remarks

The retrieved value is either the value defined by a previous call of the *SetAcceleration* method or the default value if there was no previous call of *SetAcceleration*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous Get acceleration of axis 0
double acceleration = Ch.GetAcceleration(Axis.ACSC_AXIS_0);

// Asynchronous Get acceleration of axis 0
int timeout = 2000;
ACSC_WAITBLOCK wb = Ch.GetAccelerationAsync(Axis.ACSC_AXIS_0);
double acceleration = (double)Ch.GetResult(wb, timeout);
```

3.12.7 SetDeceleration

Description

The method defines a value of motion deceleration.

Syntax

object.SetDeceleration(Axis a, double deceleration)

Async Syntax

ACSC_WAITBLOCK object.SetDecelerationAsync(Axis axis, double deceleration)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
deceleration	The value specifies a required motion deceleration. The value will be used in the subsequent motions except the master-slave motions.

Return Value

None

Remarks

The method writes the specified value to the controller. One value can be specified for each axis.

A single-axis motion uses the value of the corresponding axis. A multi-axis motion uses the value of the leading axis. The leading axis is an axis specified first in the motion Command.

The method affects the motions initiated after the method call. The method has no effect on any motion that was started or planned before the method call. To change deceleration of an executed or planned motion, use the SetDecelerationImm method.

The method has no effect on the master-slave motions.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous Set deceleration of axis 0 double deceleration = 100000;
// Writes the specified deceleration value to the controller
Ch.SetDeceleration(Axis.ACSC_AXIS_0, deceleration);
// Asynchronous Set deceleration of axis 0
ACSC_WAITBLOCK wb = Ch.SetDecelerationAsync(Axis.ACSC_AXIS_0,
deceleration);
object pWait = 0;
```

3.12.8 SetDecelerationImm

Description

The method defines a value of motion deceleration. Unlike *SetDeceleration*, the method immediately affects any executed or intended motion.

Syntax

object.SetDecelerationImm(Axis axis, double deceleration)

Async Syntax

ACSC_WAITBLOCK object.SetDecelerationImmAsync(Axis axis, double deceleration)

Arguments

Axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
Deceleration	The value specifies a required motion deceleration. The value will be applied to all motions whether in progress or to be started.

Return Value

None

Remarks

The method writes the specified value to the controller. One value can be specified for each axis.

A single-axis motion uses the value of the corresponding axis. A multi-axis motion uses the value of the leading axis. The leading axis is an axis specified first in the motion command.

The method affects:

> The currently executed motion.

> The waiting motions that were planned before the method call.

> The motions that will be commanded after the method call. The method has no effect on the master-slave motions.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous Set immediate deceleration of axis 0
double deceleration = 100000;
// Writes the specified deceleration value to the controller
Ch.SetDecelerationImm(Axis.ACSC_AXIS_0, deceleration);
// Asynchronous Set immediate deceleration of axis 0
ACSC_WAITBLOCK wb = Ch.SetDecelerationImmAsync(Axis.ACSC_AXIS_0,
deceleration);
```

3.12.9 GetDeceleration

Description

The method retrieves a value of motion deceleration.

Syntax

object.GetDeceleration(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GetDecelerationAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

Double

The method retrieves the value of the motion deceleration.

Remarks

The retrieved value is either the value defined by a previous call of the *SetDeceleration* method or the default value if *SetDeceleration* was not previously called.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous get deceleration of axis 0
double deceleration = Ch.GetDeceleration(Axis.ACSC_AXIS_0);
```

```
// Asynchronous get deceleration of axis 0
ACSC_WAITBLOCK wb = Ch.GetDecelerationAsync(Axis.ACSC_AXIS_0);
double deceleration = (double)Ch.GetResult(wb, 2000);
```

3.12.10 SetJerk

Description

The method defines a value of motion jerk.

Syntax

object.SetJerk(Axis axis, double jerk)

Async Syntax

ACSC_WAITBLOCK object.SetJerkAsync(Axis axis, double jerk)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
jerk	The value specifies a required motion jerk. The value will be used in the subsequent motions except for the master-slave motions.

Return Value

None

Remarks

The method writes the specified value to the controller. One value can be specified for each axis.

A single-axis motion uses the value of the corresponding axis. A multi-axis motion uses the value of the leading axis. The leading axis is an axis specified first in the motion command.

The method affects the motions initiated after the method call. The method has no effect on any motion that was started or planned before the method call. To change the jerk of an executed or planned motion, use the *SetJerkImm* method.

The method has no effect on the master-slave motions.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous set jerk of axis 0
Ch.SetJerk(Axis.ACSC_AXIS_0, 1000000);

// Asynchronous set jerk of axis 0
ACSC_WAITBLOCK wb = Ch.SetJerkAsync(Axis.ACSC_AXIS_0, 1000000);
```

3.12.11 SetJerkImm

Description

The method defines a value of motion jerk. Unlike *SetJerk*, the method has an immediate effect on any executed and pending motion.

Syntax

object.SetJerkImm(Axis axis, double jerk)

Async Syntax

ACSC_WAITBLOCK object.SetJerkImmAsync(Axis axis, double jerk)

Arguments

Axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
Jerk	The value specifies a required motion jerk.

Return Value

None

Remarks

The method writes the specified value to the controller. One value can be specified for each axis.

A single-axis motion uses the value of the corresponding axis. A multi-axis motion uses the value of the leading axis. The leading axis is an axis specified first in the motion command.

The method affects:

- The currently executed motion.
- The waiting motions that were planned before the method call.
- The motions that will be commanded after the method call. The method has no effect on the master-slave motions.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous set immediate jerk of axis 0
Ch.SetJerkImm(Axis.ACSC_AXIS_0, 1000000);

// Asynchronous set immediate jerk of axis 0
ACSC_WAITBLOCK wb = Ch.SetJerkImmAsync(Axis.ACSC_AXIS_0, 1000000);
```

3.12.12 GetJerk

Description

The method retrieves a value of motion jerk.

Syntax

object.GetJerk(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GetJerkAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

Double

The method retrieves the value of the motion jerk.

Remarks

The retrieved value is a value defined by a previous call of *SetJerk*, or the default value if the method was not called before.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous get jerk of axis 0
double jerk = Ch.GetJerk(Axis.ACSC_AXIS_0);

// Asynchronous get jerk of axis 0
ACSC_WAITBLOCK wb = Ch.GetJerkAsync(Axis.ACSC_AXIS_0);
double jerk = (double)Ch.GetResult(wb, 2000);
```

3.12.13 SetKillDeceleration

Description

The method defines a value of motion kill deceleration.

Syntax

object.SetKillDeceleration(Axis axis, double killDeceleration)

Async Syntax

ACSC_WAITBLOCK object.SetKillDecelerationAsync(Axis Axis, double killDeceleration)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
killDeceleration	The value specifies a required motion kill deceleration. The value will be used in the subsequent motions.

Return Value

None

Remarks

The method writes the specified value to the controller. One value can be specified for each axis.

A single-axis motion uses the value of the corresponding axis. A multi-axis motion usesthe value of the leading axis. The leading axis is an axis specified first in the motion command.

The method affects the motions initiated after the method call. The method has no effect on any motion that was started or planned before the method call.

If the method fails, the Error object is filled with the Error Description.

Example

```
 // Synchronous set motion kill deceleration of axis 0
 double killdeceleration = 100000;
 // Writes the specified kill deceleration value to
//the controller
 Ch.SetKillDeceleration(Axis.ACSC_AXIS_0, killdeceleration);
 // Asynchronous set motion kill deceleration of axis 0
 ACSC_WAITBLOCK wb = Ch.SetKillDecelerationAsync(
     Axis.ACSC_AXIS_0, killdeceleration);
```

3.12.14 SetKillDecelerationImm

Description

The method defines a value of motion kill deceleration. Unlike *SetKillDeceleration*, the method has an immediate effect on any executed and plannedmotion.

Syntax

object.SetKillDecelerationImm(Axis axis, double killDeceleration)

Async Syntax

ACSC_WAITBLOCK object.SetKillDecelerationImmAsync(Axis axis, double killDeceleration)

Arguments

Axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
KillDeceleration	The value specifies a required motion kill deceleration. The value will be used in the subsequent motions.

Return Value

None

Remarks

The method writes the specified value to the controller. One value can be specified for each axis.

A single-axis motion uses the value of the corresponding axis. A multi-axis motion uses the value of the leading axis. The leading axis is an axis specified first in the motion command.

The method affects:

- The currently executed motion.
- The waiting motions that were planned before the method call.

- The motions that will be commanded after the method call. The method has no effect on the master-slave motions.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous get motion kill deceleration of axis 0
double killDeceleration =
   Ch.GetKillDeceleration(Axis.ACSC_AXIS_0);
// Asynchronous get motion kill deceleration of axis 0
ACSC_WAITBLOCK wb =
   Ch.GetKillDecelerationAsync(Axis.ACSC_AXIS_0);
killDeceleration = (double)Ch.GetResult(wb, 2000);
```

3.12.15 GetKillDeceleration

Description

The method retrieves a value of motion kill deceleration.

Syntax

object.GetKillDeceleration(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GetKillDecelerationAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

Double

The method retrieves the value of the motion kill deceleration.

Remarks

The retrieved value is a value defined by a previous call of *SetKillDeceleration*, or the default value if the method was not previously called.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous get motion kill deceleration of axis 0
double killDeceleration = Ch.GetKillDeceleration(Axis.ACSC_AXIS_0);
// Asynchronous get motion kill deceleration of axis 0
ACSC_WAITBLOCK wb = Ch.GetKillDecelerationAsync(Axis.ACSC_AXIS_0);
killDeceleration = (double)Ch.GetResult(wb, 2000);
```

3.12.16 SetFPosition

Description

The method assigns a current value of feedback position.

Syntax

object.SetFPosition(Axis axis, double fPosition)

Async Syntax

ACSC_WAITBLOCK object.SetFPositionAsync(Axis axis, double fPosition)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
fPosition	The value specifies the current value of feedback position.

Return Value

None

Remarks

The method assigns a current value to the feedback position. No physical motion occurs. The motor remains in the same position; only the internal controller offsets are adjusted so that the periodic calculation of the feedback position will provide the required results.

For more information see the explanation of the **SET** command in the *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous set feedback position value of 0 to axis 0
Ch.SetFPosition(Axis.ACSC_AXIS_0, 0);

// Asynchronous set feedback position value of 0 to axis 0
ACSC_WAITBLOCK wb = Ch.SetFPositionAsync(Axis.ACSC_AXIS_0, 0);
```

3.12.17 GetFPosition

Description

The method retrieves the current value of the motor feedback position.

Syntax

object.GetFPosition(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GetFPositionAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

Double

The method retrieves an instant value of the motor feedback position.

Remarks

The feedback position is a measured position of the motor transferred to user units. If the method fails, the Error object is filled with the Error Description.

Example

```
 // Synchronous get motor feedback position
double fPosition = Ch.GetFPosition(Axis.ACSC_AXIS_0);
 // Asynchronous get motor feedback position
 ACSC_WAITBLOCK wb = Ch.GetFPositionAsync(Axis.ACSC_AXIS_0);
fPosition = (double)Ch.GetResult(wb, 2000);
```

3.12.18 SetRPosition

Description

The method assigns a current value of reference position.

Syntax

object.SetRPosition(Axis axis, double rPosition)

Async Syntax

ACSC_WAITBLOCK object.SetRPositionAsync(Axis axis, double rPosition)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
rPosition	The value specifies the current value of reference position.

Return Value

None

Remarks

The method assigns a current value to the reference position. No physical motion occurs. The motor remains in the same position; only the internal controller offsets are adjusted so that the periodic calculation of the reference position will provide the required results.

For more information see explanation of the **SET** command in the *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous set reference position value of 0 to axis 0
Ch.SetRPosition(Axis.ACSC_AXIS_0, 0);

// Asynchronous set reference position value of 0 to axis 0
ACSC_WAITBLOCK wb = Ch.SetRPositionAsync(Axis.ACSC_AXIS_0, 0);
```

3.12.19 GetRPosition

Description

The method retrieves an instant value of the motor reference position.

Syntax

object.GetRPosition(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GetRPositionAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

Double

The method retrieves the current value of the motor reference position.

Remarks

The method retrieves the current value of the motor reference position. The reference position is a value calculated by the controller as a reference for the motor.

If the method fails, the Error object is filled with the Error Description.

Example

```
 // Synchronous get instant value of motor reference position
double rPosition = Ch.GetRPosition(Axis.ACSC_AXIS_0);
// Asynchronous get instant value of motor reference position
ACSC_WAITBLOCK wb = Ch.GetRPositionAsync(Axis.ACSC_AXIS_0);
rPosition = (double)Ch.GetResult(wb, 2000);
```

3.12.20 GetFVelocity

Description

The method retrieves the instantaneous value of the motor feedback velocity. Unlike

GetVelocity, this method retrieves the actually measured velocity and not the required value.

Syntax

object.GetFVelocity(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GetFVelocityAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

Double

Remarks

The feedback velocity is a measured velocity of the motor transferred to user units. If the method fails, the Error object is filled with the Error Description.

Example

```
 // Synchronous get instant value of motor feedback velocity
 double fVelocity = Ch.GetFVelocity(Axis.ACSC_AXIS_0);
// Asynchronous get instant value of motor feedback velocity
 ACSC_WAITBLOCK wb = Ch.GetFVelocity(Axis.ACSC_AXIS_0);
fVelocity = (double)Ch.GetResult(wb, 2000);
```

3.12.21 GetRVelocity

Description

The method retrieves an instant value of the motor referencevelocity.

Syntax

object.GetRVelocity(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GetRVelocityAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

Double

The method retrieves the current value of the motor reference velocity.

Remarks

The reference velocity is a value calculated by the controller in the process of motion generation. If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous get instant value of motor reference velocity
double rVelocity = Ch.GetRVelocity(Axis.ACSC_AXIS_0);
// Asynchronous get instant value of motor reference velocity
ACSC_WAITBLOCK wb = Ch.GetRVelocity(Axis.ACSC_AXIS_0);
rVelocity = (double)Ch.GetResult(wb, 2000);
```

3.13 Axis/Motor Management Methods

The Axis/Motor Management methods are:

Table 3-18. Axis/Motor Management Methods

Method	Description
CommutExt	Commutates a motor.
Enable	Activates an axis.
EnableM	Activates several axes.
Disable	Shuts off an axis.
DisableAll	Shuts off all axes.
DisableExt	Shuts off an axis and defines the disable reason.
DisableM	Shuts off several axes.
Group	Creates a coordinate system for a multi-axis motion.
Split	Breaks down a previously created axis group.
SplitAll	Breaks down all previously created axis groups.

3.13.1 CommutExt

DESCRIPTION

The method initiates motor commutation.

SYNTAX

object.CommutExt(Axis axis, float current, int settle, int slope)

ASYNC SYNTAX

ACSC_WAITBLOCK object.CommutExtAsync(Axis axis, float current, int settle, int slope)

ARGUMENTS

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
current	Desired excitation current in percentage 0-100, **ACSC_NONE** for default value.
settle	Specifies the time it takes for auto commutation to settle, in milliseconds. **ACSC_NONE** for the default value of 500ms.
slope	Specifies the time it takes for the current to rise to the desired value, **ACSC_NONE** for default value.

RETURN VALUE

None

REMARKS

The method commutates a motor. After the commutation, the motor begins to follow the reference and physical motion is available.

If the method fails, the Error object is populated with the Error Description.

EXAMPLE

```
int timeout = 5000;
// Commutate axis 0
Ch.CommutExt(Axis.ACSC_AXIS_0, -1, 750, 500);
//Wait till axis 0 is commutated during 5 sec
Ch.WaitMotorCommutated(Axis.ACSC_AXIS_0,1,timeout);
```

3.13.2 Enable

Description

The method activates an axis.

Syntax

object.Enable(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.EnableAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

None

Remarks

The method activates an axis. After activation, the axis begins to follow the reference and physical motion is available.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
// Enable axis 0
Ch.Enable(Axis.ACSC_AXIS_0);
//Wait till axis 0 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0,1,timeout);
```

3.13.3 EnableM

Description

The method activates severalmotors.

Syntax

object.EnableM(Axis[] axes)

Async Syntax

ACSC_WAITBLOCK object.EnableMAsync(Axis[] axes)

Arguments

axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.

Return Value

None

Remarks

The method activates several motors. After the activation, the motors begin to follow the corresponding reference and physical motions for the specified motors are available.

If the method fails, the Error object is filled with the Error Description.

Example

```
// create axes array, terminate with Axis.ACSC_NONE
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,Axis.ACSC_NONE };
// Enable of axes 0 and 1
Ch.EnableM(axes);
```

3.13.4 Disable

Description

The method shuts off an axis.

Syntax

object.Disable(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.DisableAsync(Axis axis)

Arguments

Axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

None

Remarks

The method shuts off a motor. After shutting off the motor cannot follow the reference and remains at idle.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
// Disable axis 0
Ch.Disable(Axis.ACSC_AXIS_0);
//Wait for motor 0 to disable for 5 sec.
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0,0,timeout);
```

3.13.5 DisableAll

Description

The method shuts off all axes.

Syntax

object.DisableAll()

Async Syntax

ACSC_WAITBLOCK object.DisableAllAsync()

Arguments

None

Return Value

None

Remarks

The method shuts off all motors. After the shutting off none of motors can follow the corresponding, reference and all motors remain idle.

If no motors are currently enabled, the method has no effect.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Disable all motors
Ch.DisableAll();
```

3.13.6 DisableExt

Description

The method shuts off an axis and defines the disable reason.

Syntax

object.DisableExt(Axis axis, int reason)

Async Syntax

ACSC_WAITBLOCK object.DisableExtAsync(Axis axis, int reason)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
reason	Integer number that defines the reason of disabling the axis. The specified value is stored in the **MERR** variable in the controller and so modifies the state of the disabled motor.

Return Value

None

Remarks

The method shuts off a motor. After shutting off the motor cannot follow the reference and remains at idle.

If **reason** specifies one of the available motor termination codes, the state of the disabled motor will be identical to the state of the motor disabled for the corresponding fault. This provides an enhanced implementation of user-defined fault response.

If the second parameter specifies an arbitrary number, the motor state will be displayed as "Kill/disable reason: <number> - customer code". This provides the ability to separate different **DISABLE** commands in the application.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
int myCode = 10;
// Disable axis 0 with internal customer code
Ch.DisableExt(Axis.ACSC_AXIS_0, myCode);
//Wait for motor 0 to disable for 5 sec.
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0,0,timeout);
```

3.13.7 DisableM

Description

The method shuts off several axes.

Syntax

object.DisableM(Axis[] axes)

Async Syntax

ACSC_WAITBLOCK object.DisableMAsync(Axis[] axes)

Arguments

axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.

Return Value

None

Remarks

The method shuts off several axes. After the shutting off, the axes cannot follow the corresponding reference and remain idle.

If the method fails, the Error object is filled with the Error Description.

Example

```
// create axes array, terminate with Axis.ACSC_NONE
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1, Axis.ACSC_NONE };
// Disable of axes 0 and 1
Ch.DisableM(axes);
```

3.13.8 Group

Description

The method creates a coordinate system for a multi-axis motion.

Syntax

object.Group(Axis[] axes)

Async Syntax

ACSC_WAITBLOCK object.GroupAsync(Axis[] axes)

Arguments

axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.

Return Value

None

Remarks

The method creates a coordinate system for a multi-axis motion. The first element of the **axes** array specifies the leading axis. The motion parameters of the leading axis become the default motion parameters for the group.

An axis can belong toonly one group at a time. If the application requires restructuring the axes, it must split the existing group and only then create the new one. To split the existing group, use Split.To split all existing groups, use *SplitAll*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// create axes array, terminate with Axis.ACSC_NONE
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1, Axis.ACSC_AXIS_2,
Axis.ACSC_NONE };
// Create group of axes 0, 1 and 2.
Ch.Group(axes);
```

3.13.9 Split

Description

The method breaks down a previously created axis group.

Syntax

object.Split(Axis[] axes)

Async Syntax

object.Split(Axis[] axes)

Arguments

axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.

Return Value

None

Remarks

The method breaks down an axis group created before by the *Group* method. **axes** must specify the same axes as for the **Group** method that created the group. After the splitting up, the group no longer exists.

If the method fails, the Error object is filled with the Error Description.

Example

```
// create axes array, terminate with Axis.ACSC_NONE
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1, Axis.ACSC_AXIS_2,
Axis.ACSC_NONE };
// Create group of axes 0, 1 and 2.
Ch.Group(axes);
// Split the group of axes 0, 1 and 2.
Ch.Split(axes);
```

3.13.10 SplitAll

Description

The method breaks down all axis groups created before.

Syntax

object.SplitAll()

Async Syntax

ACSC_WAITBLOCK object.SplitAllAsync()

Arguments

None

Return Value

None

Remarks

The method breaks down all axis groups created before by the *Group* method.

The application may call this method to ensure that no axes are grouped. If no groups currently exist, the method has no effect.

If the method fails, the Error object is filled with the Error Description.

Example

```
// create axes array, terminate with Axis.ACSC_NONE
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1, Axis.ACSC_AXIS_2,
Axis.ACSC_NONE };
// Create group of axes 0, 1 and 2.
Ch.Group(axes);
Ch.SplitAll(); // Split all groups created before
```

3.14 Motion Management Methods

The Motion Management methods are:

Table 3-19. Motion Management Methods

Method	Description
Go	Starts up a motion that is waiting in the specified motion queue.
GoM	Synchronously starts up several motions that are waiting in the specified motion queues.
Halt	Terminates a motion using the full deceleration profile.
HaltM	Terminates several motions using the full deceleration profile.
Kill	Terminates a motion using the reduced deceleration profile.
KillAll	Terminates all currently executed motions.
KillM	Terminates several motions using the reduced deceleration profile.
KillExt	Terminates a motion using reduced deceleration profile and defines the kill reason.
Break	Terminates a motion immediately and provides a smooth transition to the next motion.
BreakM	Terminates several motions immediately and provides a smooth transition to the next motions.

3.14.1 Go

Description

The method starts up a motion waiting in the specified motion queue.

Syntax

object.Go(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GoAsync(Axis axis)

Arguments

Axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

None

Remarks

A motion that was set with the **ACSC_AMF_WAIT** flag does not start until the **Go** method is executed.The motion waits in the appropriate motion queue.

Each axis has a separate motion queue. A single-axis motion waits in the motion queue of the corresponding axis. A multi-axis motion waits in the motion queue of the leading axis. The leading axis is an axis specified first in the motion command.

The **Go** method initiates the motion waiting in the motion queue of the specified axis. If no motion waits in the motion queue, the method has no effect.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
// Enable axis 0
Ch.Enable(Axis.ACSC_AXIS_0);
// Wait till axis 0 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0,1,timeout);
// Wait till GO command executed on axis 0,target position of 10000
Ch.ToPoint(MotionFlags.ACSC_AMF_WAIT, Axis.ACSC_AXIS_0,10000);
// Motion Start
Ch.Go(Axis.ACSC_AXIS_0);
// Finish the motion, End of multi-point motion
Ch.EndSequence(Axis.ACSC_AXIS_0);
```

3.14.2 GoM

Description

The method synchronously starts up several motions that are waiting in the specified motion queues.

Syntax

object.GoM(Axis[] axes)

Async Syntax

ACSC_WAITBLOCK object.GoMAsync(Axis[] axes)

Arguments

Axes	Array of axis constants: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc.. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.

Return Value

None

Remarks

Axes motions that were set with the **ACSC_AMF_WAIT** flag do not start until the **GoM** method is executed. The motions wait in the appropriate motion queue.

Each axis has a separate motion queue. A single-axis motion waits in the motion queue of the corresponding axis. A multi-axis motion waits in the motion queue of the leading axis. The leading axis is an axis specified first in the motion command.

The **GoM** method initiates the motions waiting in the motion queues of the specified axes. If no motion waits in one or more motion queues, the corresponding axes are not affected.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
    Axis.ACSC_NONE };
double[] points = { 10000, 10000 };
Ch.EnableM(axes); // Enable axes 0 and 1

// Wait till axis 0 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Wait till axis 1 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);

// Wait till GO GUI executed on axes 0 and 1 target position
// of 10000,0000
Ch.ToPointM(MotionFlags.ACSC_AMF_WAIT, axes, points);
// Start the motion of 0 and 1
Ch.GoM(axes);
// Finish the motion, end of the multi-point motion
Ch.EndSequenceM(axes);
```

3.14.3 Halt

Description

The method terminates a motion using the full deceleration profile.

Syntax

object.Halt(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.HaltAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

None

Remarks

The method terminates the executed motion that involves the specified axis. The terminated motion can be either single-axis or multi-axis. Any other motion waiting in the corresponding motion queue is discarded and will not be executed.

If no executed motion involves the specified axis, the method has no effect.

The terminated motion finishes using the full third-order deceleration profile and the motion deceleration value.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Ch.Enable(Axis.ACSC_AXIS_0); // Enable axes 0
// Wait till axis 0 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Relative motion of axis 0,target position of 10000
Ch.ToPoint(MotionFlags.ACSC_AMF_RELATIVE, Axis.ACSC_AXIS_0,
    10000);
// Halt executed to axis 0
Ch.Halt(Axis.ACSC_AXIS_0);
// Finish the motion
// End of the motion
Ch.EndSequence(Axis.ACSC_AXIS_0);
```

3.14.4 HaltM

Description

The method terminates several motions using the full decelerationprofile.

Syntax

object.HaltM(Axis[] axes)

Async Syntax

ACSC_WAITBLOCK object.HaltMAsync(Axis[] axes)

Arguments

axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.

Return Value

None

Remarks

The method terminates all executed motions that involve the specified axes. The terminated motions can be either single-axis or multi-axis. All other motions waiting in the corresponding motion queues are discarded and will not be executed.

If no executed motion involves a specified axis, the method has no effect on the corresponding axis.

The terminated motions finish using the full third-order deceleration profile and the motion deceleration values.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
 Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
     Axis.ACSC_NONE };
 double[] points = { 10000, 10000 };
 Ch.EnableM(axes); // Enable axes 0 and 1
 // Wait till axis 0 is enabled during 5 sec
 Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
 // Wait till axis 1 is enabled during 5 sec
 Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
 // Relative motion of axes 0 and 1 with target position of
 // 10000,10000
 Ch.ToPointM(MotionFlags.ACSC_AMF_RELATIVE, axes, points);
 // Halt executed on axes 0 and 1
 Ch.HaltM(axes);
 // Finish the motion
 // End of the multi-point motion
 Ch.EndSequenceM(axes);
```

3.14.5 Kill

Description

The method terminates a motion using reduced deceleration profile.

Syntax

object.Kill(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.KillAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

None

Remarks

The method terminates the executed motion that involves the specified axis. The terminated motion can be either single-axis or multi-axis. Any other motion waiting in the corresponding motion queue is discarded and will not be executed.

If no executed motion involves the specified axis, the method has no effect.

The terminated motion finishes with the reduced second-order deceleration profile and the kill deceleration value.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Ch.Enable(Axis.ACSC_AXIS_0);
// Enable axes 0
// Wait till axis 0 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Relative motion of axis 0,target position of 10000
Ch.ToPoint(MotionFlags.ACSC_AMF_RELATIVE, Axis.ACSC_AXIS_0,
    10000);
// Kill axis 0
Ch.Kill(Axis.ACSC_AXIS_0);
// Finish the motion
// End of the motion
Ch.EndSequence(Axis.ACSC_AXIS_0);
```

3.14.6 KillAll

Description

The method terminates all currently executed motions.

Syntax

object.KillAll()

Async Syntax

ACSC_WAITBLOCK object.KillAllAsync()

Arguments

None

Return Value

None

Remarks

The method terminates all currently executed motions. Any other motion waiting in any motion queue is discarded and will not be executed.

If no motion is currently executed, the method has no effect.

The terminated motions finish with the reduced second-order deceleration profile and the kill deceleration values.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
    Axis.ACSC_NONE };
double[] points = { 10000, 10000 };
Ch.EnableM(axes); // Enable axes 0 and 1
                  // Wait till axis 0 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Wait till axis 1 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
// Relative motion of axes 0 and 1 with target position of
// 10000,10000
Ch.ToPointM(MotionFlags.ACSC_AMF_RELATIVE, axes, points);
// Kill axes 0 and 1
Ch.KillAll();
// Finish the motion
// End of the multi-point motion
Ch.EndSequenceM(axes);
```

3.14.7 KillM

Description

The method terminates several motions using reduced deceleration profile.

Syntax

object.KillM(Axis[] axes)

Async Syntax

ACSC_WAITBLOCK object.KillMAsync(Axis[] axes)

Arguments

axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.

Return Value

None

Remarks

The method terminates all executed motions that involve the specified axes. The terminated motions can be either single-axis or multi-axis. All other motions waiting in the corresponding motion queues are discarded and will not be executed.

If no executed motion involves a specified axis, the method has no effect on the corresponding axis.

The terminated motions finish with the reduced second-order deceleration profile and the kill deceleration values.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
    Axis.ACSC_NONE };
double[] points = { 10000, 10000 };
Ch.EnableM(axes); // Enable axes 0 and 1
// Wait till axis 0 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Wait till axis 1 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
// Relative motion of axes 0 and 1 with target position of
// 10000,10000
Ch.ToPointM(MotionFlags.ACSC_AMF_RELATIVE, axes, points);
// Kill axes 0 and 1
Ch.KillM(axes);
// Finish the motion
// End of the multi-point motion
Ch.EndSequenceM(axes);
```

3.14.8 KillExt

Description

The method terminates a motion using reduced deceleration profile and defines the kill reason.

Syntax

object.KillExt(Axis axis, int reason)

Async Syntax

ACSC_WAITBLOCK object.KillExtAsync(Axis axis, int reason)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
reason	Integer number that defines the reason of disabling the axis. The specified value is stored in the **MERR** variable in the controller and so modifies the state of the disabled motor.

Return Value

None

Remarks

The method terminates the executed motion that involves the specified axis. The terminated motion can be either single-axis or multi-axis. Any other motion waiting in the corresponding motion queue is discarded and will not be executed.

If no executed motion involves the specified axis, the method has no effect.

The terminated motion finishes with the reduced second-order deceleration profile and the kill deceleration value.

If **reason** specifies one of the available motor termination codes, the state of the killed motor will be identical to the state of the motor killed for the corresponding fault. This provides an enhanced implementation of user-defined fault response.

If the second parameter specifies an arbitrary number, the motor state will be displayed as "Kill/disable reason: <number> - customer code". This provides ability to separate different Kill commands in the application.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
int myCode = 10;
Ch.Enable(Axis.ACSC_AXIS_0); // Enable axes 0
// Wait till axis 0 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Relative motion of axis 0,target position of 10000
Ch.ToPoint(MotionFlags.ACSC_AMF_RELATIVE, Axis.ACSC_AXIS_0,
    10000);
// Kill axis 0 with internal customer code
Ch.KillExt(Axis.ACSC_AXIS_0, myCode);
// Finish the motion
// End of the motion
Ch.EndSequence(Axis.ACSC_AXIS_0);
```

3.14.9 Break

Description

Terminates a motion immediately and provides a smooth transition to the nextmotion.

Syntax

object.Break(Axis axis)

Async Syntax

object.BreakAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

None

Remarks

The method terminates the executed motion that involves the specified axis only if the next motion is waiting in the corresponding motion queue. The terminated motion can be either single-axis or multi-axis.

If the motion queue contains no waiting motion, the break command is not executed immediately. The current motion continues instead until the next motion is planned to the same motion queue. Only then is the break command executed.

If no executed motion involves the specified axis, or the motion finishes before the next motion is planned, the method has noeffect.

When executing the break command, the controller terminates the motion immediately without any deceleration profile. The controller builds instead a smooth third-order transition profile to the next motion.

Use caution when implementing the break command with a multi-axis motion, because the controller provides a smooth transition profile of the vector velocity. In a single-axis motion, this ensures a smooth axis velocity. However, in a multi-axis motion an axis velocitycan change abruptly if the terminated and next motions are not tangent in the junction point. To avoid jerk, the terminated and next motion must be tangent or nearly tangent in thejunction point.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Ch.Enable(Axis.ACSC_AXIS_0); // Enable axes 0
// Wait till axis 0 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Start up the motion of axis 0 to point 10000
Ch.ToPoint(MotionFlags.ACSC_AMF_RELATIVE, Axis.ACSC_AXIS_0,
    10000);
```

```
// Executing of break to axis 0
Ch.Break(Axis.ACSC_AXIS_0);
// Change the end of the point to point 0 of the fly
Ch.ToPoint(MotionFlags.ACSC_AMF_RELATIVE, Axis.ACSC_AXIS_0,
    0);
// Finish the motion
// End of the motion
Ch.EndSequence(Axis.ACSC_AXIS_0);
```

3.14.10 BreakM

Description

Terminates several motions immediately and provides a smooth transition to the next motions.

Syntax

object.BreakM(Axis[] axes)

Async Syntax

ACSC_WAITBLOCK object.BreakMAsync(Axis[] axes)

Arguments

axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.

Return Value

None

Remarks

The method terminates the executed motions that involve the specified axes. Only those motions are terminated that have the next motion waiting in the corresponding motion queue. The terminated motions can be either single-axis or multi-axis.

If a motion queue contains no waiting motion, the break command does not immediately affect the corresponding axis. The current motion continues instead until the next motion is planned to the same motion queue. Only then, the break command is executed.

If no executed motion involves the specified axis, or the corresponding motion finishes before the next motion is planned, the method does not affect the axis.

When executing the break command, the controller terminates the motion immediately without any deceleration profile. Instead, the controller builds a smooth third-order transition profile to the next motion.

Use caution when implementing the break command with a multi-axis motion, because the controller provides a smooth transition profile of the vector velocity. In a single-axis motion, this ensures a smooth axis velocity, but in a multi-axis motion, an axis velocity can change abruptly if the terminated and next motions are not tangent in the junction point. To avoid jerk, the terminated and next motion must be tangent or nearly tangent in the junction point.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
    Axis.ACSC_NONE };
double[] points = { 10000, 10000 };
Ch.EnableM(axes); // Enable axes 0 and 1
// Wait till axis 0 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Wait till axis 1 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
// Start up motion of axes 0 and 1 to point of 10000
Ch.ToPointM(MotionFlags.ACSC_AMF_RELATIVE, axes, points);
// Execute break to axes 0 and 1
Ch.BreakM(axes);
// Change the end point to -10000 on the fly
points[0] = -10000; points[1] = -10000;
Ch.ToPointM(MotionFlags.ACSC_AMF_RELATIVE, axes, points);
// Finish the motion
// End of the multi-point motion
Ch.EndSequenceM(axes);
```

3.15 Point-to-Point Motion Methods

The Point-to-Point Motion methods are:

Table 3-20. Point-to-Point Motion Methods

Method	Description
ToPoint	Initiates a single-axis motion to the specified point.
ToPointM	Initiates a multi-axis motion to the specified point.
ExtToPoint	Initiates a single-axis motion to the specified point using the specified velocity or end velocity.
ExtToPointM	Initiates a multi-axis motion to the specified point using the specified velocity or end velocity.

3.15.1 ToPoint

Description

The method initiates a single-axis motion to the specified point.

Syntax

object.ToPoint(MotionFlags flags, Axis axis, double point)

Async Syntax

ACSC_WAITBLOCK object.ToPointAsync(MotionFlags flags, Axis axis, double point)

Arguments

flags	Bit-mapped parameter that can include one or more of the following flags: **ACSC_AMF_WAIT**: plan the motion but do not start it until the Go method is executed. **ACSC_AMF_RELATIVE**: the **Point** value is relative to the end point of the previous motion. If the flag is not specified, the Point specifies an absolute coordinate.
axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
point	Coordinate of the target point.

Return Value

None

Remarks

The method initiates a single-axis point-to-point motion.

The motion is executed using the required motion velocity and finishes with zero end velocity. The required motion velocity is the velocity specified by the previous call of the SetVelocity method or the default velocity if the method was not called.

To execute multi-axis point-to-point motion, use *ToPointM*. To execute motion with other motion velocity or non-zero end velocity, use *ExtToPoint* or *ExtToPointM*.

The controller response indicates that the command was accepted and the motion was planned successfully. The method does not wait for the motion end.

To wait for the motion end, use *WaitMotionEnd* method.

The motion builds the velocity profile using the required values of velocity, acceleration, deceleration, and jerk of the specified axis.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Ch.Enable(Axis.ACSC_AXIS_0); // Enable axes 0
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Relative motion of axis 0 to target point of 10000
Ch.ToPoint(MotionFlags.ACSC_AMF_RELATIVE, Axis.ACSC_AXIS_0,
```

```
                 10000);
    // Wait till motion ends
    Ch.WaitMotionEnd(Axis.ACSC_AXIS_0, timeout * 2);
    // Finish the motion
```

3.15.2 ToPointM

Description

The method initiates a multi-axis motion to the specified point.

Syntax

object.ToPointM(MotionFlags flags, Axis[] axes, double[] point)

Async Syntax

ACSC_WAITBLOCK object.ToPointMAsync(MotionFlags flags, Axis[] axes, double[] point)

Arguments

flags	Bit-mapped parameter that can include one or more of the following flags: **ACSC_AMF_WAIT**: plan the motion but do not start it until the GoM method is executed. **ACSC_AMF_RELATIVE**: the Point value is relative to the end point of the previous motion. If the flag is not specified, the Point specifies an absolute coordinate. **ACSC_AMF_MAXIMUM**: not to use the motion parameters from the leading axis but to calculate the maximum allowed motion velocity, acceleration, deceleration, and jerk of the involved axes.
axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.
point	Array of the target coordinates. The number and order of values must correspond to the **axes** array. The **point** must specify a value for each element of **axes** except the last –1 element.

Return Value

None

Remarks

The method initiates a multi-axis point-to-point motion.

The motion is executed using the required motion velocity and finishes with zero end velocity. The required motion velocity is the velocity specified by the previous call of the *SetVelocity* method, or the default velocity if the method was not called.

To execute single-axis point-to-point motion, use *ToPoint*. To execute motion with other motion velocity or non-zero end velocity, use *ExtToPoint* or *ExtToPointM*.

The controller response indicates that the command was accepted and the motion was planned successfully. The method does not wait for the motion end. To wait for the motion end, use *WaitMotionEnd* method.

The motion builds the velocity profile using the required values of velocity, acceleration, deceleration, and jerk of the leading axis. The leading axis is the first axis in the **Axes** array.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
    Axis.ACSC_NONE };
double[] points = { 10000, 10000 };
Ch.EnableM(axes); // Enable axes 0 and 1
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Wait till axis 1 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
// Immediately start the motion of the axes X,Y to the
// absolute target points
Ch.ToPointM(MotionFlags.ACSC_NONE, axes, points);
// Finish the motion
// End of the multi-point motion
Ch.EndSequenceM(axes);
```

3.15.3 ExtToPoint

Description

The method initiates a single-axis motion to the specified point using the specified velocity or end velocity.

Syntax

object.ExtToPoint(MotionFlags flags, Axis axis, double point, double velocity, double endVelocity)

Async Syntax

ACSC_WAITBLOCK object.ExtToPointAsync(MotionFlags flags, Axis axis, double point, double Velocity, double EndVelocity)

Arguments

flags	Bit-mapped parameter that can include one or more of the following flags: **ACSC_AMF_WAIT**: plan the motion but don't start it until the method Go is executed. **ACSC_AMF_RELATIVE**: the Point value is relative to the end point of the previous motion. If the flag is not specified, the Point specifies an absolute coordinate. **ACSC_AMF_VELOCITY**: the motion will use velocity specified by the Velocity argument instead of the default velocity. **ACSC_AMF_ENDVELOCITY**: the motion will come to the end point with the velocity specified by the EndVelocity argument.
axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
point	Coordinate of the target point.
velocity	Motion velocity. The argument is used only if the **ACSC_AMF_VELOCITY** flag is specified.
endVelocity	Velocity in the target point. The argument is used only if the **ACSC_AMF_ENDVELOCITY** flag is specified. Otherwise, the motion finishes with zero velocity.

Return Value

None

Remarks

The method initiates a single-axis point-to-point motion.

If the **ACSC_AMF_VELOCITY** flag is specified, the motion is executed using the velocity specified by **velocity**. Otherwise, the required motion velocity is used. The required motion velocity is the velocity specified by the previous call of *SetVelocity*, or the default velocity if the method was not called.

If the **ACSC_AMF_ENDVELOCITY** flag is specified, the motion velocity at the final point is specified by the **endVelocity**. Otherwise, the motion velocity at the final point is zero.

To execute a multi-axis point-to-point motion with the specified velocity or end velocity, use *ExtAddPointM*. To execute motion with default motion velocity and zero end velocity, use *ToPoint* or *ToPointM*.

The controller response indicates that the command was accepted and the motion was planned successfully. The method does not wait for the motion end. To wait for the motion end, use *WaitMotionEnd*.

The motion builds the velocity profile using the required values of acceleration,deceleration and jerk of the specified axis.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Ch.Enable(Axis.ACSC_AXIS_0); // Enable axis 0
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Parameters :
// Ch.ACSC_AMF_VELOCITY Or 'Ch.ACSC_AMF_ENDVELOCITY -
// Start up the motion with specified velocity '5000
// Come to the end point with specified 'velocity 1000
// Ch.ACSC_AXIS_0 - axis 0
// 10000 - target point
// 5000 - motion velocity
// 1000 - velocity in the target point
Ch.ExtToPoint(
MotionFlags.ACSC_AMF_VELOCITY |
    MotionFlags.ACSC_AMF_ENDVELOCITY, Axis.ACSC_AXIS_0,
    10000, 5000, 1000);
// Wait motion motion during 20 sec
Ch.WaitMotionEnd(Axis.ACSC_AXIS_0, timeout * 4);
// Finish the motion
// End of the multi-point motion
Ch.EndSequence(Axis.ACSC_AXIS_0);
```

3.15.4 ExtToPointM

Description

The method initiates a multi-axis motion to the specified point using the specified velocity or end velocity.

Syntax

object.ExtToPointM(MotionFlags flags, Axis[] axes, double[] point, double velocity, double endVelocity)

Async Syntax

ACSC_WAITBLOCK object.ExtToPointMAsync(MotionFlags flags, Axis[] axes, double[] point, double velocity, double endVelocity)

Arguments

flags	Bit-mapped parameter that can include one or more of the following flags: **ACSC_AMF_WAIT**: plan the motion but don't start it until the method GoM is executed. **ACSC_AMF_RELATIVE**: the Point value is relative to the end point of the previous motion. If the flag is not specified, the Point specifies an absolute coordinate. **ACSC_AMF_VELOCITY**: the motion will use velocity specified by the Velocity argument instead of the default velocity. **ACSC_AMF_ENDVELOCITY**: the motion will come to the end point with the velocity specified by the EndVelocity argument.
axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.
point	Array of the target coordinates. The number and order of values must correspond to the **axes** array. The **point** must specify a value for each element of **axes** except the last –1 element.
velocity	Motion velocity. The argument is used only if the **ACSC_AMF_VELOCITY** flag is specified.
endVelocity	Velocity in the target point. The argument is used only if the **ACSC_AMF_ENDVELOCITY** flag is specified. Otherwise, the motion finishes with zero velocity.

Return Value

None

Remarks

The method initiates a multi-axis point-to-point motion.

If the **ACSC_AMF_VELOCITY** flag is specified, the motion is executed using the velocity specified by velocity. Otherwise, the required motion velocity is used. The required motion velocity is the velocity specified by the previous call of *SetVelocity*, or the default velocity if the method was not called.

If the **ACSC_AMF_ENDVELOCITY** flag is specified, the motion velocity at the final point is specified by the **endVelocity**. Otherwise, the motion velocity at the final point is zero.

To execute a single-axis point-to-point motion with the specified velocity or end velocity, use *ExtToPoint*. To execute a motion with default motion velocity and zero end velocity, use *ToPoint* or *ToPointM*.

The controller response indicates that the command was accepted and the motion was planned successfully. The method does not wait for the motion end. To wait for the motion end, use the WaitMotionEnd method.

The motion builds the velocity profile using the required values of acceleration, deceleration and jerk of the leading axis. The leading axis is the first axis in the **axes** array.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
 Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
     Axis.ACSC_NONE };
 double[] points = { 1000, 2000 };
 Ch.EnableM(axes); // Enable axis 0 and 1
 // Wait axis 0 enabled during 5 sec
 Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
 // Wait till axis 1 is enabled during 5 sec
 Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
 // Parameters :
 // Ch.ACSC_AMF_VELOCITY Or 'Ch.ACSC_AMF_ENDVELOCITY
 // Start up the motion with specified velocity '5000
 // Come to the end point with specified 'velocity 1000
 // axes - axis 0 and 1
 // points - target points
 // 5000 - motion velocity
 // 1000 - velocity in the target point
 Ch.ExtToPointM(
 MotionFlags.ACSC_AMF_VELOCITY |
     MotionFlags.ACSC_AMF_ENDVELOCITY,
     axes, points, 5000, 1000);
 // Finish the motion
 // End of the multi-point motion
 Ch.EndSequenceM(axes);
```

3.16 Track Motion Control Method

The Track Motion Control method is:

Table 3-21. Track Motion Control Method

Method	Description
Track	The method initiates a single-axis track motion.

3.16.1 Track

Description

The method initiates a single-axis track motion.

Syntax

object.Track(MotionFlags flags, Axis axis)

Async Syntax

ACSC_WAITBLOCK object.TrackAsync(MotionFlags flags, Axis axis)

Arguments

flags	Bit-mapped parameter that can include one or more of the following flag: **ACSC_AMF_WAIT**: plan the motion but do not start it until the Go method is executed.
axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

None

Remarks

The method initiates a single-axis track motion. After the motion is initialized, PTP motion will be generated with every change in the **TPOS** value.

The controller response indicates that the command was accepted and the motion was planned successfully.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Ch.Enable(Axis.ACSC_AXIS_0); // Enable axis 0
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Start up immediately the motion of the axis 0
Ch.Track(MotionFlags.ACSC_NONE, Axis.ACSC_AXIS_0);
// Target position to 5000
Ch.Transaction("TPOS0 = 5000");
```

3.17 Jog Methods

The Jog methods are:

Table 3-22. Jog Methods

Method	Description
Jog	Initiates a single-axis jog motion.
JogM	Initiates a multi-axis jog motion.

3.17.1 Jog

Description

The method initiates a single-axis jog motion.

Syntax

object.Jog(MotionFlags flags, Axis axis, double velocity)

Async Syntax

ACSC_WAITBLOCK object.JogAsync(MotionFlags flags, Axis axis, double velocity)

Arguments

flags	Bit-mapped parameter that can include one or more of the following flags: **ACSC_AMF_WAIT**: plan the motion but don't start it until the method Go is executed. **ACSC_AMF_VELOCITY**: the motion will use velocity specified by the Velocity argument instead of the default velocity.
axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
velocity	**If the ACSC_AMF_VELOCITY** flag is specified, the velocity profile is built using the value of **velocity**. The sign of **velocity** defines the direction of the motion. If the **ACSC_AMF_VELOCITY** flag is not specified, only the sign of velocity is used in order to specify the direction of motion. In this case, the constants **GlobalDirection.ACSC_POSITIVE_DIRECTION or GlobalDirection.ACSC_NEGATIVE_DIRECTION** can be used.

Return Value

None

Remarks

The method initiates a single-axis jog. To execute multi-axis jog, use *JogM*.

The jog motion is a motion with constant velocity and no defined ending point. The jog motion continues until the next motion is planned, or the motion is killed for any reason.

The motion builds the velocity profile using the default values of acceleration, deceleration and jerk of the specified axis. If the **ACSC_AMF_VELOCITY** flag is not specified, the default value of **velocity** is used as well. In this case, only the sign of velocity is used in order to specify the direction of motion. The positive velocity defines a positive direction, the negative velocity defines the negative direction.

If the **ACSC_AMF_VELOCITY** flag is specified, the value of **velocity** is used instead of the default velocity. The sign of **velocity** defines the direction of the motion.

The method waits for the controller response.

The controller response indicates that the command was accepted and the motion was planned successfully. No waiting for the motion end is provided.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Ch.Enable(Axis.ACSC_AXIS_0); // Enable axis 0
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Initiates a single-axis jog motion to axis 0 to
// positive direction with
// the specified velocity 10000
Ch.Jog(MotionFlags.ACSC_NONE, Axis.ACSC_AXIS_0,
     (double)GlobalDirection.ACSC_NEGATIVE_DIRECTION);
```

3.17.2 JogM

Description

The method initiates a multi-axis jog motion.

Syntax

object.JogM(MotionFlags flags, Axis[] axes, GlobalDirection[] direction, double velocity)

Async Syntax

ACSC_WAITBLOCK object.JogMAsync(MotionFlags flags, Axis[] axes, GlobalDirection[] direction, double velocity)

Arguments

flags	Bit-mapped parameter that can include one or more of the following flags: **ACSC_AMF_WAIT**: plan the motion but don't start it until the method GoM is executed. **ACSC_AMF_VELOCITY**: the motion will use velocity specified by the Velocity argument instead of the default velocity.
axes	Array of axis constants. Each element specifies one involved axis: ACSC_ AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.
direction	Array of directions - The number and order of values must correspond to the **axes** array. The **direction** array must specify direction for each element of axes except the last –1 element. The value **ACSC_POSITIVE_DIRECTION** in the **direction** array specifies the correspondent axis to move in positive direction, the value **ACSC_NEGATIVE_DIRECTION** specifies the correspondent axis to move in the negative direction.

velocity	If the **ACSC_AMF_VELOCITY** flag is specified, the velocity profile is built using the value of velocity. If the **ACSC_AMF_VELOCITY** flag is not specified, **velocity** is not used.

Return Value

None

Remarks

The method initiates a multi-axis jog motion. To execute single-axis jog motion, use *Jog*.

The jog motion is a motion with constant velocity and no defined ending point. The jog motion continues until the next motion is planned, or the motion is killed for anyreason.

The motion builds the vector velocity profile using the default values of velocity, acceleration, deceleration and jerk of the axis group. If the **ACSC_AMF_VELOCITY** flag is not specified, the default value of velocity is used as well. If the **ACSC_AMF_VELOCITY** flag is specified, the value of **velocity** is used instead of the default velocity.

The method waits for the controller response.

The controller response indicates that the command was accepted and the motion was planned successfully. The method cannot wait or validate the end of the motion. To wait for the motion end, use *WaitMotionEnd*.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
Axis.ACSC_NONE };
GlobalDirection[] directions = {
    GlobalDirection.ACSC_POSITIVE_DIRECTION,
    GlobalDirection.ACSC_POSITIVE_DIRECTION };
Ch.EnableM(axes); // Enable axes 0 and 1
                  // Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Wait till axis 1 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
// Start up immediately the jog motion of axes X,Y to
// positive directions
// with the specified velocity 5000
Ch.JogM(MotionFlags.ACSC_NONE, axes, directions, 5000);
// Finish the motion
// End of the multi-point motion
Ch.EndSequenceM(axes);
```

3.18 Slaved Motion Methods

The Slaved Motion methods are:

Table 3-23. Slaved Motion Methods

Method	Description
SetMaster	Initiates calculation of a master value for an axis.
Slave	Initiates a master-slave motion.
SlaveStalled	Initiates master-slave motion with limited following area.

3.18.1 SetMaster

Description

The method initiates calculating of master value for an axis.

Syntax

object.SetMaster(Axis axis, stringformula)

Async Syntax

ACSC_WAITBLOCK object.SetMasterAsync(Axis axis, stringformula)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
formula	ASCII string that specifies a rule for calculating master value.

Return Value

None

Remarks

The method initiates calculating of master value for an axis.

The master value for each axis is presented in the controller as one element of the **MPOS** array. Once the **SetMaster** method is called, the controller is calculates the master value forthe specified axis each controller cycle.

The **SetMaster** method can be called again for the same axis at any time. At that moment, the controller discards the previous formula and accepts the newly specified formula for the master calculation.

The method waits for the controller response.

The controller response indicates that the command was accepted and the controller starts calculating the master value according to the formula.

The **formula** string can specify any valid ACSPL+ expression that uses any ACSPL+ or user global variables as itsoperands.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
// Master value is calculated as feedback position of the
// axis 1 with scale factor equal 2
string mFormula = "2 * FPOS(1)";
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
    Axis.ACSC_NONE };
// Enable axes
Ch.EnableM(axes);
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Wait till axis 1 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
// Set master formula "FPOS(0)= 2 * FPOS(1)"
Ch.SetMaster(Axis.ACSC_AXIS_0, mFormula);
//Set axis 0 to slave
Ch.Slave(MotionFlags.ACSC_NONE, Axis.ACSC_AXIS_0);
// Relative motion with target position of 10000
Ch.ToPoint(MotionFlags.ACSC_AMF_RELATIVE, Axis.ACSC_AXIS_1,
    10000);
// Finish the motion
// End of the multi-point motion
Ch.EndSequenceM(axes);
```

3.18.2 Slave

Description

The method initiates a master-slave motion.

Syntax

object.Slave(MotionFlasg flags, Axis axis)

Async Syntax

ACSC_WAITBLOCK object.SlaveAsync(MotionFlags flags, Axis axis)

Arguments

flags	Bit-mapped parameter that can include one or more of the following flags: **ACSC_AMF_WAIT:** plan the motion but don’t start it until the method Go is executed. **ACSC_AMF_POSITIONLOCK**: the motion will use position lock. If the flag is not specified, velocity lock is used (see *Remarks*).
axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

None

Remarks

The method initiates a single-axis master-slave motion with an unlimited area of following. If the area of following must be limited, use *SlaveStalled*.

The master-slave motion continues until the motion is killed or the motion fails for any reason. The method waits for the controller response.

The controller response indicates that the command was accepted and the motion was planned successfully.

The master value for the specified axis must be defined before by the call to **SetMaster** method. The **SetMaster** method can be called again in order to change the formula of master calculation. If at this moment the master-slave motion is in progress, the slave can come out from synchronism. The controller then regains synchronism, probably with a different value of offset between the master and slave.

If the ACSC_AMF_POSITIONLOCK flag is not specified, the method activates a velocity- lock mode of slaved motion. When synchronized, the **APOS** axis reference follows the **MPOS** with a constant offset:

APOS = MPOS + C

The value of **C** is latched at the moment when the motion comes to synchronism, and then remains unchanged as long as the motion is synchronous. If at the moment of motion start the master velocity is zero, the motion starts synchronously and C is equal to the difference between initial values of **APOS** and **MPOS**.

If the **ACSC_AMF_POSITIONLOCK** flag is specified, the method activates a position-lock mode of slaved motion. When synchronized, the **APOS** axis reference strictly follows the **MPOS**:

APOS = MPOS

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
// Master value is calculated as feedback position
// of theaxis 1 with scale factor equal 2
string mFormula = "2 * FPOS(1)";
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
    Axis.ACSC_NONE };
// Enable axes
Ch.EnableM(axes);
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Wait till axis 1 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
// Set master formula "FPOS(0)= 2 * FPOS(1)"
Ch.SetMaster(Axis.ACSC_AXIS_0, mFormula);
//Set axis 0 to slave
Ch.Slave(MotionFlags.ACSC_NONE, Axis.ACSC_AXIS_0);
// Relative motion with target position of 10000
Ch.ToPoint(MotionFlags.ACSC_AMF_RELATIVE, Axis.ACSC_AXIS_1,
```

```
        10000);
    // Finish the motion
    // End of the multi-point motion
    Ch.EndSequenceM(axes);
```

3.18.3 SlaveStalled

Description

The method initiates master-slave motion within predefined limits.

Syntax

object.SlaveStalled(MotionFlasg flags, Axis axis, double left, double right)

Async Syntax

ACSC_WAITBLOCK object.SlaveStalledAsync(MotionFlags flags, Axis axis, double left, double right)

Arguments

flags	Bit-mapped parameter that can include one or more of the following flags: **ACSC_AMF_WAIT**: plan the motion but don’t start it until the method Go is executed. **ACSC_AMF_POSITIONLOCK**: the motion will use position lock. If the flag is not specified, velocity lock is used (see *Remarks*).
axis	Axis constant of slaved axis: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
left	Left (negative) limit of the axis.
right	Right (positive) limit of the axis.

Return Value

None

Remarks

The method initiates single-axis master-slave motion within predefined limits. Use Slave to initiate unlimited motion. For sophisticated forms of master-slave motion, use slaved variants of segmented and spline motions.

The method waits for the controller response.

The controller response indicates that the command was accepted and the motion was planned successfully.

The master-slave motion continues until the *Kill* command is executed, or the motion fails for any reason.

The master value for the specified axis must be defined before by the call to *SetMaster* method. The *SetMaster* method can be called again in order to change the formula of master calculation. If at this moment the master-slave motion is in progress, the slave can come out from synchronism. The controller then regains synchronism, probably with adifferent value of offset between the master and slave.

If the ACSC_AMF_POSITIONLOCK flag is not specified, the method activates a velocity- lock mode of slaved motion. When synchronized, the **APOS** axis reference follows the **MPOS** with a constant offset:

APOS = MPOS + C

The value of **C** is latched at the moment when the motion comes to synchronism, and then remains unchanged as long as the motion is synchronous. If at the moment of motion start the master velocity is zero, the motion starts synchronously and **C** is equal to thedifference between initial values of **APOS** and **MPOS**.

If the ACSC_AMF_POSITIONLOCK flag is specified, the method activates a position-lock mode of slaved motion. When synchronized, the **APOS** axis reference strictly follows the **MPOS**:

APOS = MPOS

The **left** and **right** values define the allowed area of changing the **APOS** value. The **MPOS** value is not limited and can exceed the limits. In this case, the motion comes out from synchronism, and the **APOS** value remains (stalls) in one of the limits until the change of **MPOS** allows following again.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
// Master value is calculated as feedback position
// of the axis 1 with scale factor equal 2
string mFormula = "2 * FPOS(1)";
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
    Axis.ACSC_NONE };
// Enable axes
Ch.EnableM(axes);
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Wait till axis 1 is enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
// Set master formula "FPOS(0)= 2 * FPOS(1)"
Ch.SetMaster(Axis.ACSC_AXIS_0, mFormula);
// Left (negative) limit of the following area, right
// (positive)limit of the following area
Ch.SlaveStalled(MotionFlags.ACSC_NONE, Axis.ACSC_AXIS_0,
    -5000, 5000);
// Relative motion with target position of 10000
Ch.ToPoint(MotionFlags.ACSC_AMF_RELATIVE, Axis.ACSC_AXIS_1,
    10000);
// Finish the motion
// End of the multi-point motion
Ch.EndSequenceM(axes);
```

3.19 Multi-Point Motion Methods

The Multi-Point Motion methods are:

Table 3-24. Multi-Point Motion Methods

Method	Description
MultiPoint	Initiates a single-axis multi-point motion.
MultiPointM	Initiates a multi-axis multi-point motion.

3.19.1 MultiPoint

Description

The method initiates a single-axis multi-point motion.

Syntax

object.MultiPoint(MotionFlags flags, Axis axis, double dwell)

Async Syntax

ACSC_WAITBLOCK object.MultiPointAsync(MotionFlags flags, Axis axis, double dwell)

Arguments

flags	Bit-mapped parameter that can include one or more of the following flags: **ACSC_AMF_WAIT**: plan the motion but don't start it until the method Go is executed. **ACSC_AMF_RELATIVE**: the coordinates of each point are relative. The first point is relative to the instant position when the motion starts; the second point is relative to the first, etc. If the flag is not specified, the coordinates of each point are absolute. **ACSC_AMF_VELOCITY**: the motion uses the velocity specified with each point instead of the default velocity. **ACSC_AMF_CYCLIC**: the motion uses the point sequence as a cyclic array. After positioning to the last point it does positioning to the first point and continues.
axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
dwell	Dwell in each point in milliseconds.

Return Value

None

Remarks

The method initiates a single-axis multi-point motion. To execute multi-axis multi-point motion, use *MultiPointM*.

The motion executes sequential positioning to each of the specified points, optionally with dwell in each point.

The method itself does not specify any point, so that the created motion starts only after the first point is specified. The points of motion are specified by using the *AddPoint* or *ExtAddPoint* methods that follow this method.

The motion finishes when the *EndSequence* method is executed. If the **EndSequence** call is omitted, the motion will stop at the last point of the sequence and wait for the next point. No transition to the next motion in the motion queue will occur until the method **EndSequence** executes.

The method waits for the controller response.

The controller response indicates that the command was accepted and the motion was planned successfully. The method cannot wait or validate the end of the motion. To wait for the motion end, use the *WaitMotionEnd*.

During positioning to each point, a velocity profile is built using the default values of acceleration, deceleration, and jerk of the specified axis. If the ACSC_AMF_VELOCITY flag is not specified, the default value of velocity is used as well. If the ACSC_AMF_VELOCITY flag is specified, the value of velocity specified in subsequent ExtAddPoint methods is used.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Ch.Enable(Axis.ACSC_AXIS_0); // Enable axis 0
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
//Create multi-point motion of axis 0 with default velocity
// with dwell 1 ms
Ch.MultiPoint(MotionFlags.ACSC_NONE, Axis.ACSC_AXIS_0, 1);
// Add some points
for (int index = 0; index < 5; index++)
{
    Ch.AddPoint(Axis.ACSC_AXIS_0, 100.0 * index);
}
// Finish the motion
// End of the multi-point motion
Ch.EndSequence(Axis.ACSC_AXIS_0);
```

3.19.2 MultiPointM

Description

The method initiates a multi-axis multi-point motion.

Syntax

object.MultiPointM(MotionFlags flags, Axis[] axes, double dwell)

Async Syntax

ACSC_WAITBLOCK object.MultiPointMAsync(MotionFlags flags, Axis[] axes, double dwell)

Arguments

flags	Bit-mapped parameter that can include one or more of the following flags: ACSC_AMF_WAIT: plan the motion but don’t start it until the method GoM is executed. ACSC_AMF_RELATIVE: the coordinates of each point are relative. The first point is relative to the instant position when the motion starts; the second point is relative to the first, etc. If the flag is not specified, the coordinates of each point are absolute. ACSC_AMF_VELOCITY: the motion uses the velocity specified with each point instead of the default velocity. ACSC_AMF_CYCLIC: the motion uses the point sequence as a cyclic array. After positioning to the last point it does positioning to the first point and continues.
axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.
dwell	Dwell in each point in milliseconds.

Return Value

None

Remarks

The method initiates a multi-axis multi-point motion. To execute single-axis multi-point motion, use *MultiPoint*.

The motion executes sequential positioning to each of the specified points, optionally with dwell in each point.

The method itself does not specify any point, so the created motion starts only after the first point is specified. The points of motion are specified by using *AddPointM* or *ExtAddPointM*, methods that follow this method.

The motion finishes when the *EndSequenceM* method is executed. If the call of **EndSequenceM** is omitted, the motion will stop at the last point of the sequence and wait for the next point. No transition to the next motion in the motion queue will occur until the method **EndSequenceM** executes.

The method waits for the controller response.

The controller response indicates that the command was accepted and the motion was planned successfully. The method cannot wait or validate the end of the motion. To wait for the motion end, use the *WaitMotionEnd* method.

During positioning to each point, a vector velocity profile is built using the default values of velocity, acceleration, deceleration, and jerk of the axis group. If the **AFM_VELOCITY** flag is not specified, the default value of velocity is used as well. If the **AFM_VELOCITY** flag is specified, the value of velocity specified in subsequent ExtAddPointM methods isused.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
 Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
     Axis.ACSC_NONE };
 double[] points = { 0, 0 };
 Ch.EnableM(axes); // Enable axis 0 and 1
                   // Wait axis 0 enabled during 5 sec
 Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
 // Wait axis 1 enabled during 5 sec
 Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
 // Create multi-point motion of axis 0 and 1 with default
 // velocity without
 // dwell in the points
 Ch.MultiPointM(MotionFlags.ACSC_NONE, axes, 0);
 // Add some points
 for (int index = 0; index < 5; index++)
 {
     points[0] = 100 * index;
     points[1] = 100 * index;
     Ch.AddPointM(axes, points);
 }
 // Finish the motion
 // End of the multi-point motion
 Ch.EndSequenceM(axes);
```

3.20 Arbitrary Path Motion Methods

The Arbitrary Path Motion methods are:

Table 3-25. Arbitrary Path Motion Methods

Mehtod	Description
Spline	Initiates a single-axis spline motion. The motion follows an arbitrary path defined by a set of points.
SplineM	Initiates a multi-axis spline motion. The motion follows an arbitrary path defined by a set of points.

3.20.1 Spline

Description

The method initiates a single-axis spline motion. The motion follows an arbitrary path defined by a set of points.

Syntax

object.Spline(MotionFlags flags, Axis axis, [double period])

Async Syntax

ACSC_WAITBLOCK object.SplineAsync(MotionFlags flags, Axis axis, [double period])

Arguments

flags	Bit-mapped parameter that can include one or more of the following flags: **ACSC_AMF_WAIT**: plan the motion but don't start it until the method Go is executed. **ACSC_AMF_RELATIVE**: the coordinates of each point are relative. The first point is relative to the instant position when the motion starts; the second point is relative to the first, etc. If the flag is not specified, the coordinates of each point are absolute. **ACSC_AMF_VARTIME**: the time interval between adjacent points is non-uniform and is specified along with each added point. If the flag is not specified, the interval is uniform and is specified in the Period argument. **ACSC_AMF_CYCLIC**: use the point sequence as a cyclic array: after the last point come to the first point and continue. **ACSC_AMF_CUBIC**: use a cubic interpolation between the specified points (third-order spline).
Axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
period	Time interval between adjacent points. The parameter is used only if the **ACSC_AMF_VARTIME** flag is not specified.

Return Value

None

Remarks

The method initiates a single-axis spline motion. To execute multi-axis spline motion, use *SplineM*.

The method waits for the controller response.

The controller response indicates that the command was accepted and the motion was planned successfully. The method cannot wait or validate the end of the motion. To wait for the motion end, use the *WaitMotionEnd* method.

The motion does not use the default values of velocity, acceleration, deceleration, and jerk. The points and the time intervals between the points completely define the motion profile.

Points for arbitrary path motion are defined by the consequent calls of *AddPoint* or *ExtAddPoint* methods. The *EndSequence* method terminates the point sequence. After execution of the *EndSequence* method, no *AddPoint* or *ExtAddPoint* methods for this motion are allowed.

If **flag** is not specified, linear interpolation is used (first-order spline).

Time intervals between the points are uniform, or non-uniform as defined by the **ACSC_AMF_VARTIME** flag. If **ACSC_AMF_VARTIME** is not specified, the controller builds PV spline motion.

If **ACSC_AMF_VARTIME** is specified, the controller builds PVT spline motion. The trajectory of the motion follows through the defined points. Each point presents the instant desired position at a specific moment.

This motion does not use a motion profile generation. The time intervals between the points are typically short, so that the array of the points implicitly specifies the desired velocity in each point.

If the time interval does not coincide with the controller cycle, the controller provides interpolation of the points according to the **ACSC_AMF_CUBIC** flag.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
double[] points = { 0, 0 };
Ch.Enable(Axis.ACSC_AXIS_0); // Enable axis 0
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Create the arbitrary path motion to axis 0, use a cubic
// interpolation between the specified points with uniform
// interval 500 ms
Ch.Spline(MotionFlags.ACSC_AMF_CUBIC, Axis.ACSC_AXIS_0, 500);
// Add some points
for (int index = 0; index < 5; index++)
{
    points[0] = 100 * index;
    points[1] = 20000 * index;
    Ch.AddPVPoint(Axis.ACSC_AXIS_0, points[0], points[1]);
}
// Finish the motion
// End of the multi-point motion
Ch.EndSequence(Axis.ACSC_AXIS_0);
```

3.20.2 SplineM

Description

The method initiates a multi-axis spline motion. The motion follows an arbitrary path defined by a set of points.

Syntax

object.SplineM(MotionFlags flags, Axis[] axes, [double period])

Async Syntax

ACSC_WAITBLOCK object.SplineMAsync(MotionFlags flags, Axis[] axes, [double period])

Arguments

flags	Bit-mapped parameter that can include one or more of the following flags: **ACSC_AMF_WAIT**: plan the motion but don't start it until the method GoM is executed. **ACSC_AMF_RELATIVE**: the coordinates of each point are relative. The first point is relative to the instant position when the motion starts; the second point is relative to the first, etc. If the flag is not specified, the coordinates of each point are absolute. **ACSC_AMF_VARTIME**: the time interval between adjacent points is non-uniform and is specified along with each added point. If the flag is not specified, the interval is uniform and is specified in the **Period** argument. **ACSC_AMF_CYCLIC**: use the point sequence as a cyclic array: after the last point come to the first point and continue. **ACSC_AMF_CUBIC**: use a cubic interpolation between the specified points (third-order spline).
axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants Axis Definitions.
period	Time interval between adjacent points. The parameter is used only if the **ACSC_AMF_VARTIME** flag is not specified.

Return Value

None

Remarks

The method initiates a multi-axis spline motion. To execute a single-axis spline motion, use Spline.

The method waits for the controller response.

The controller response indicates that the command was accepted and the motion was planned successfully. The method cannot wait or validate the end of the motion. To wait for the motion end, use the WaitMotionEnd method.

The motion does not use the default values of velocity, acceleration, deceleration, and jerk. The points and the time intervals between the points define the motion profile completely.

Points for arbitrary path motion are defined by the consequent calls of the *AddPVpointM* or *ExtAddPointM* methods. The *EndSequenceM* method terminates the point sequence. After execution of the EndSequenceM method, no *AddPointM* or *ExtAddPointM* methods for this motion are allowed.

The trajectory of the motion follows through the defined points. Each point presents the instant desired position at a specific moment. Time intervals between the points are uniform, or non-uniform as defined by the **ACSC_AMF_VARTIME** flag.

This motion does not use motion profile generation. Typically, the time intervals between the points are short, so that the array of the points implicitly specifies the desired velocity in each point.

If the time interval does not coincide with the controller cycle, the controller provides interpolation of the points according to the **ACSC_AMF_CUBIC** flag.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
    Axis.ACSC_NONE };
double[] points = { 0, 0 };
Ch.EnableM(axes); // Enable axes 0 and 1
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Wait axis 1 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
// Create the arbitrary path motion to axes 0 and 1 with
// uniform interval 10 ms use a cubic interpolation
// between the specified points
Ch.SplineM(MotionFlags.ACSC_AMF_CUBIC, axes, 10);
// Add some points
for (int index = 0; index < 5; index++)
{
    points[0] = 100 * index;
    points[1] = 200 * index;
    Ch.AddPVPointM(axes, points, points);
}
// Finish the motion
// End of the multi-point motion
Ch.EndSequenceM(axes);
```

3.21 PVT Methods

Table 3-26. PVT Methods

Method	Description
AddPVPoint	Adds a point to a single-axis multi-point or spline motion.
AddPVPointM	Adds a point to a multi-axis multi-point or spline motion.
AddPVTPoint	Adds a point to a single-axis multi-point or spline motion.
AddPVTPointM	Adds a point to a multi-axis multi-point or spline motion.

3.21.1 AddPVPoint

Description

The method adds a point to a single-axis PV spline motion and specifies velocity.

Syntax

ACSC_WAITBLOCK object.AddPVPoint(Axis axis, double point, double velocity)

Async Syntax

object.AddPVPointAsync(Axis axis, double point, double velocity)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
point	Coordinate of the added point.
velocity	Desired velocity at the point

Return Value

None

Remarks

Before this method can be used, initiate PV spline motion by calling Spline with the appropriate flags.

The method adds a point to a single-axis PV spline motion with a uniform time and specified velocity at that point

To add a point to a multi-axis PV motion, use AddPVTPointM and ExtAddPointM.

To add a point to a PVT motion with non-uniform time interval, use the AddPVTPoint and AddPVTPointM methods. The method waits for the controller response.

The controller response indicates that the command was accepted and the point is added to the motion buffer. The point can be rejected if the motion buffer is full. In this case, call this method periodically until the method returns a non-zero value.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Create PV motion with uniform time interval with uniform
// interval 500 ms
int timeout = 5000;
double point = 0;
double velocity = 0;
Ch.Enable(Axis.ACSC_AXIS_0); // Enable axis 0
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
Ch.Spline(MotionFlags.ACSC_AMF_CUBIC, Axis.ACSC_AXIS_0, 500);
// Add some points
```

```
for (int index = 0; index < 5; index++)
{
    // Position and velocity for each point
    point = 10 * index;
    velocity = 20 * index;
    Ch.AddPVPoint(Axis.ACSC_AXIS_0, point, velocity);
}
// Finish the motion
// End of the multi-point motion
Ch.EndSequence(Axis.ACSC_AXIS_0);
```

3.21.2 AddPVPointM

Description

The method adds a point to a multiple-axis PV spline motion and specifies velocity.

Syntax

ACSC_WAITBLOCK object.AddPVPointM(Axis[] axes, double[] point, double[] velocity)

Async Syntax

object.AddPVPointMAsync(Axis[] axes, double[] point, double[] velocity)

Arguments

axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.
point	Array of the coordinates of added point. The number and order of values must correspond to the **axes** array. **point** must specify a value for each element of **axes** except the last –1 element.
velocity	Array of the velocities of added point. The number and order of values must correspond to the **axes** array. The **velocity** must specify a value for each element of **axes** except the last –1 element.

Return Value

None

Remarks

Before this method can be used, PVT spline motion must be initiated by calling *SplineM* with the appropriate flags.

The method adds a point to a multiple-axis PV spline motion with a uniform time and specified velocity at that point.

To add a point to a single-axis PV motion, use AddPVPoint. To add a point to a PVT motion with non-uniform time interval, use the AddPVTPoint and AddPVTPointM methods.

The method waits for the controller response.

The controller response indicates that the Commandwas accepted and the point is added to the motion buffer. The point can be rejected if the motion buffer is full. In this case, you can call this method periodically until the method returns non-zero value.

All axes specified in the **Axes** array must be specified before calling the *MultiPointM* or the *SplineM* method. The number and order of the axes in the **Axes** array must correspond exactly to the number and order of the axes of *MultiPointM* or the *SplineM* methods.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
Axis.ACSC_NONE };
double[] points = { 0, 0 };
double[] velocity = { 0, 0 };
Ch.EnableM(axes); // Enable axes 0 and 1
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Wait axis 1 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
// Create PV motion with uniform time interval with uniform
// interval 1000 ms
Ch.SplineM(MotionFlags.ACSC_AMF_CUBIC, axes, 1000);
// Add some points
for (int index = 0; index < 5; index++)
{
    // Position and velocity for each point
    points[0] = 10 * index;
    points[1] = points[0];
    velocity[0] = 20 * index;
    velocity[1] = velocity[0];
    Ch.AddPVPointM(axes, points, velocity);
}
// Finish the motion
// End of the multi-point motion
Ch.EndSequenceM(axes);
```

3.21.3 AddPVTPoint

Description

The method adds a point to a single-axis PVT spline motion and specifies velocity and motion time.

Syntax

object.AddPVTPoint(Axis axis, double point, double velocity, [double timeInterval])

Async Syntax

ACSC_WAITBLOCK object.AddPVTPointAsync(Axis axis, double point, double velocity, [double timeInterval])

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
point	Coordinate of the added point.
velocity	Desired velocity at the point
timeInterval	If the motion was activated by the Spline method with the **ACSC_AMF_VARTIME** flag, this parameter defines the time interval between the previous point and the present one.

Return Value

None

Remarks

Before this method can be used, PV spline motion must be initiated by calling *Spline* with the appropriate flags.

The method adds a point to a single-axis PVT spline motion with a non-uniform time and specified velocity at that point.

To add a point to a multi-axis PVT motion, use *AddPVPointM*. To add a point to a PV motion with uniform time interval, use the *AddPVPoint* and *AddPVPointM* methods.

The method waits for the controller response.

The controller response indicates that the command was accepted and the point is added to the motion buffer. The point can be rejected if the motion buffer is full. In this case, you can call this method periodically until the method returns non-zero value.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
double point = 0;
double velocity = 0;
Ch.Enable(Axis.ACSC_AXIS_0); // Enable axis 0
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// PVT motion and uniform interval is not used
Ch.Spline(MotionFlags.ACSC_AMF_CUBIC |
MotionFlags.ACSC_AMF_VARTIME, Axis.ACSC_AXIS_0, 0);
// Add some points
for (int index = 0; index < 5; index++)
{
    point = 10 * index;
    velocity = 20 * index;
    // Position and velocity and time interval of 500 ms for
    // each point
```

```
    Ch.AddPVTPoint(Axis.ACSC_AXIS_0, point, velocity,
    500);
}
// Finish the motion
// End of the multi-point motion
Ch.EndSequence(Axis.ACSC_AXIS_0);
```

3.21.4 AddPVTPointM

Description

The method adds a point to a multiple-axis PVT spline motion and specifies velocity and motion time.

Syntax

object.AddPVTPointM(Axis[] axes, double[] point, double[] velocity, [double timeInterval])

Async Syntax

ACSC_WAITBLOCK object.AddPVTPointMAsync(Axis[] axes, double[] point, double[] velocity, [double timeInterval])

Arguments

axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.
point	Array of the coordinates of added point. The number and order of values must correspond to the **axes** array. The **point** must specify a value for each element of **axes** except the last –1 element.
velocity	Array of the velocities of added point. The number and order of values must correspond to the **axes** array. The **velocity** must specify a value for each element of **axes** except the last –1 element.
timeInterval	If the motion was activated by the Spline method with the **ACSC_AMF_VARTIME** flag, this parameter defines the time interval between the previous point and the present one.

Return Value

None

Remarks

Before this method can be used, PVT spline motion must be initiated by calling *SplineM* with the appropriate flags.

The method adds a point to a multiple-axis PVT spline motion with a non-uniform time and specified velocity at that point.

To add a point to a single-axis PVT motion, use *AddPVPoint*. To add a point to a PV motion with a uniform time interval, use the *AddPVPoint* and *AddPVPointM* methods.

The method waits for the controller response.

The controller response indicates that the command was accepted and the point is added to the motion buffer. The point can be rejected if the motion buffer is full. In this case, you can call this method periodically until the method returns non-zero value.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
Axis.ACSC_NONE };
double[] points = { 0, 0 };
double[] velocity = { 0, 0 };
Ch.EnableM(axes); // Enable axes 0 and 1
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Wait axis 1 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
// PVT motion and uniform interval is not used
Ch.SplineM(MotionFlags.ACSC_AMF_CUBIC |
MotionFlags.ACSC_AMF_VARTIME, axes, 0);
// Add some points
for (int index = 0; index < 5; index++)
{
    points[0] = 10 * index;
    points[1] = points[0];
    velocity[0] = 20 * index;
    velocity[1] = velocity[0];
    Ch.AddPVTPointM(axes, points, points, 1000);
}
// Finish the motion
// End of the multi-point motion
Ch.EndSequenceM(axes);
```

3.22 Segmented Motion Methods

The Segmented Motion methods are:

Table 3-27. Segmented Motion Methods

Method	Description
SegmentLine	Adds a linear segment to a segmented motion.
SegmentLineV2	Adds a linear segment that starts at the current point and ends at the destination point of segmented motion.

Method	Description
ExtendedSegmentArc1	Adds an arc segment to a segmented motion and specifies the coordinates of center point, coordinates of the final point, and the direction of rotation.
ExtendedSegmentArc2	Adds an arc segment to a segmented motion and specifies the coordinates of center point and rotation angle.
Stopper	Provides a smooth transition between two segments of segmented motion.
Projection	Sets a projection matrix for a segmented motion.

3.22.1 ExtendedSegmentedMotionV2

This function replaces ExtendedSegmentMotionExt, which is now obsolete.

Description

The function initiates a multi-axis extended segmented motion.

Syntax

void acsc_ExtendedSegmentedMotionExtV2(MotionFlags Flags, Axis [] Axes, double [] Point, double Velocity, double EndVelocity, double JunctionVelocity, double Angle, double CurveVelocity, double Deviation, double Radius, double MaxLength, double StarvationMargin, string Segments, int ExtLoopType, double MinSegmentLength, double MaxAllowedDeviation, int OutputIndex, int BitNumber, int Polarity, double MotionDelay);

Arguments

Flags	Bit-mapped argument that can include one or more of the following flags: **ACSC_AMF_WAIT** - plan the motion but do not start it until the function **acsc_GoM** is executed. **ACSC_AMF_VELOCITY** - the motion will use velocity specified for each segment instead of the default velocity. **ACSC_AMF_ENDVELOCITY** - This flag requires an additional parameter that specifies end velocity. The controller decelerates to the specified velocity in the end of segment. The specified value should be less than the required velocity; otherwise the parameter is ignored. This flag affects only one segment. This flag also disables corner detection and processing at the end of segment. If this flag is not specified, deceleration is not required. However, in special cases the deceleration might occur due to corner processing or other velocity control conditions. **ACSC_AMF_MAXIMUM** - use maximum velocity under axis limits. With this suffix, no required velocity should be specified. The required velocity is calculated for each segment individually on the base of segment geometry and axis velocities (VEL values) of the involved axes. **ACSC_AMF_JUNCTIONVELOCITY** - Decelerate to corner. This flag requires additional parameter that specifies corner velocity. The controller detects corner on the path and decelerates to the specified velocity before the corner. The specified value should be less than the required velocity; otherwise the parameter is ignored. If the **ACSC_AMF_JUNCTIONVELOCITY** flag is not specified while the **ACSC_AMF_ANGLE** flag is specified, the value of corner velocity is assumed to be zero. If either the **ACSC_AMF_JUNCTIONVELOCITY** nor the **ACSC_AMF_ANGLE** flags are specified, the controller provides automatic calculation as described in Automatic corner processing. **ACSC_AMF_ANGLE** - Do not treat junction as a corner, if junction angle is less than or equal to the specified value in radians.

	This flag requires additional parameter that specifies negligible angle in radians. If **ACSC_AMF_ANGLE** flag is not specified while the **ACSC_AMF_JUNCTIONVELOCITY** flag is specified, the controller accepts default value of 0.01 radians, or approximaely 0.57°. If neither the **ACSC_AMF_JUNCTIONVELOCITY** or the **ACSC_AMF_ANGLE** flags are specified, the controller provides automatic calculation as described in Automatic corner processing. **ACSC_AMF_AXISLIMIT** - Enable velocity limitations under axis limits. With this flag set, setting the **ACSC_AMF_VELOCITY** flag will result in the requested velocity being restrained by the velocity limits of all involved axes. **ACSC_AMF_CURVEVELOCITY** - Decelerate to curvature discontinuity point. This flag requires an additional parameter that specifies velocity at curvature discontinuity points. Curvature discontinuity occurs in linear-to-arc or arc-to-arc smooth junctions. If the flag is not set, the controller does not decelerate to smooth junction disregarding curvature discontinuity in the junction. If the flag is set, the controller detects curvature discontinuity points on the path and provides deceleration to the specified velocity. The specified value should be less than the required velocity; otherwise the parameter is ignored. The flag can be set together with flags **ACSC_AMF_JUNCTIONVELOCITY** and/or **ACS_AMF_ANGLE**. If neither **ACSC_AMF_JUNCTIONVELOCITY**, **ACS_AMF_ANGLE** or **ACSC_AMF_CURVEVELOCITY** is set, the controller provides automatic calculation of the corner processing. **ACSC_AMF_CURVEAUTO** - If this Flag is specified the controller provides automatic calculations as described in Enhanced automatic corner and curvature discontinuity points processing.

	ACSC_AMF_CORNERDEVIATION - Use a corner rounding option with the specified permitted deviation. This flag requires an additional parameter that specifies maximal allowed deviation of motion trajectory from the corner point. This flag cannot be set together with flags ACSC_AMF_CORNERRADIUS and ACSC_AMF_CORNERLENGTH. ACSC_AMF_CORNERRADIUS Use a corner rounding option with the specified permitted curvature. This flag requires an additional parameter that specifies maximal allowed rounding radius of the additional segment. This flag cannot be specified together with flags ACSC_AMF_CORNERLENGTH or ACSC_AMF_CORNERDEVIATION. ACSC_AMF_CORNERLENGTH Use automatic corner rounding option. This flag requires an additional parameter that specifies the maximum length of the segment for automatic corner rounding. If a length of one of the segments that built a corner exceeds the specified maximal length, the corner will not be rounded. The automatic corner rounding is only applied to pair of linear segments. If one of the segments in a pair is an arc, the rounding is not applied for this corner. This flag cannot be set together with flags ACSC_AMF_CORNERDEVIATION or ACSC_AMF_CORNERRADIUS. ACSC_AMF_EXT_LOOP Use external loops at corners. The switch requires additional parameters that specify the external loop type, the minimum segment length, and the maximum allowed deviation from profile. ACSC_AMF_EXT_LOOP_SYNC Defines output bit to support external loop synchronization.. The switch requires additional parameters that specify the output index, the bit number, and the polarity. ACSC_AMF_DELAY_MOTION Defines actual motor movement delay in microseconds. The delay resolution is 50 microseconds. The switch requires additional parameter that specify the motion delay.

	ACSC_AMF_LOCALCS Interpret entered coordinates according to the Local Coordinate System. With this switch, use 2 axes motion coordinate only (X,Y). Using 3 or more coordinates causes a runtime error.
Axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to axis 0, ACSC_AXIS_1 to axis 1, etc. After the last axis, one additional element must be located that contains –1 which marks the end of the array. For the axis constants see Axis Definitions.
Point	Array of the starting point coordinates. The number and order of values must correspond to the Axes array. Point must specify a value for each element of Axes except the last –1 element.
Velocity	If ACSC_AMF_VELOCITY flag was specified, this argument specifies a motion velocity for current segment. Set this argument to ACSC_NONE if not used.
EndVelocity	If ACSC_AMF_ENDVELOCITY flag was specified, this argument defines required velocity at the end of the current segment. Set this argument to ACSC_NONE if not used.
JunctionVelocity	If ACSC_AMF_JUNCTIONVELOCITY flag was specified, this argument defines the required velocity at the junction. Set this argument to ACSC_NONE if not used.
Angle	If ACSC_AMF_ANGLE flag was specified, this argument specifies the threshold above which a junction angle will be treated as a corner. Set this argument to ACSC_NONE if not used.
CurveVelocity	If ACSC_AMF_CURVEVELOCITY flag has been specified, this argument defines the required velocity at curvature discontinuity points. Set this argument to ACSC_NONE if not used.
Deviation	If ACSC_AMF_CORNERDEVIATION flag has been specified, this argument defines the maximal allowed trajectory deviation from the corner point. Set this argument to ACSC_NONE if not used.

Radius	If ACSC_AMF_CORNERRADIUS flag has been specified, this argument defines the maximal allowed rounding radius of the additional segment. Set this argument to ACSC_NONE if not used.
MaxLength	If ACSC_AMF_CORNERLENGTH flag has been specified, this argument defines the maximum length of the segment for processing automatic corner rounding. If a segment's corner attempts to exceed the maximum length, the corner will not be rounded. Set this argument to ACSC_NONE if not used.
StarvationMargin	Starvation margin in milliseconds. The controller sets the AST.#NEWSEGM bit to the StarvationMargin millisecond before the starvation condition occurs. Set this argument to ACSC_NONE if not used. By default, if this argument is not specified, the starvation margin is 500 milliseconds.
If the Segments parameter is used, then the starvation margin must also be defined.	
Segments	Character string that contains the name of a one-dimensional user-defined array used to store added segments. Set this argument to NULL if not used. By default, if this argument is not specified, the controller allocates internal buffer for storing 50 segments only. The argument allows the user application to reallocate the buffer for storing a larger number of segments. The larger number of segments may be required if the application needs to add many very small segments in advanced. For most applications, the internal buffer size is enough and should not be enlarged. The buffer is for the controller internal use only and should not be used by the user application. The buffer size calculation rule: each segment requires about 600 bytes, so if it is necessary to allocate the buffer for 200 segments, it should be at least 600 * 200 = 120,000 bytes. The following declaration defines a 120,000 bytes buffer: real buf (15000) See XARRSIZE explanation in the ACSPL+ Command and Variable Reference Guide for details on how to declare a buffer with more than 100,000 elements.

ExtLoopType	If ACSC_AMF_EXT_LOOP flag has been specified, this argument defines the external loop type. 0 - Cancel external loop 1 – Smooth External loop (line-arc-line) 2 – Triangle External loop (line-line-line) Set this argument to ACSC_NONE if not used
MinSegmentLength	If ACSC_AMF_EXT_LOOP flag has been specified, this argument defines the Minimum Segment Length. If the lengths of both segments are more than this value, the skywriting algorithm will be applied. Set this argument to ACSC_NONE if not used
MaxAllowedDeviation	If ACSC_AMF_EXT_LOOP flag has been specified, this argument defines the maximum allowed deviation. The parameter limits the external loop deviation from the defined profile. If the value is negative there is no limitation. Set this argument to ACSC_NONE if not used
OutputIndex	If ACSC_AMF_EXT_LOOP_SYNC flag has been specified, this argument defines the output index, the index of the digital output port to assigned to synchronization (read by OUT) Set this argument to ACSC_NONE if not used
BitNumber	If ACSC_AMF_EXT_LOOP_SYNC flag has been specified, this argument defines the bit number. Bit number (BIT_NUMBER_MIN(0) - BIT_NUMBER_MAX(16)) assigned to synchronization output. Set this argument to ACSC_NONE if not used.
Polarity	If ACSC_AMF_EXT_LOOP_SYNC flag has been specified, this argument defines the polarity. POLARITY_ON(0) or POLARITY_OFF(1) - which value is considered the initial state Set this argument to ACSC_NONE if not used

MotionDelay	If ACSC_AMF_DELAY_MOTION flag has been specified, this argument defines the actual motor movement delay in microseconds. The delay resolution is 50 microseconds. The maximum delay is 100 controller cycles or 100ms for CTIME=1ms or 20ms for CTIME=0.2ms. Set this argument to ACSC_NONE if not used.

3.22.2 SegmentLineV2

Description

The function adds a linear segment that starts at the current point and ends at the destination point of segmented motion.

Syntax

object.SegmentLineExtV2(MotionFlags Flags, Axis[] Axes, Double[] Point, double Velocity, double EndVelocity, double Time, string Values, string Variables, int Index, string Masks, int ExtLoopType, double MinSegmentLength, double MaxAllowedDeviation, int LciState);

Arguments

Flags	Bit-mapped argument that can include one or more of the following flags: ACSC_AMF_VELOCITY: the motion will use velocity specified for each segment instead of the default velocity. ACSC_AMF_ENDVELOCITY: this flag requires additional parameter that specifies end velocity. The controller decelerates to the specified velocity in the end of segment. The specified value should be less than the required velocity; otherwise the parameter is ignored. This flag affects only one segment. ACSC_AMF_VARTIME: this flag requires an addition parameter that specifies the segment processing time in milliseconds. The segment processing time defines velocity at the current segment only and has no effect on subsequent segments. This flag also disables corner detection and processing at the end of segment. If this flag is not specified, deceleration is not required. However, in special cases the deceleration might occur due to corner processing or other velocity control conditions. ACSC_AMF_USERVARIABLES: synchronize user variables with segment execution. This flag requires additional parameters that specify values, user variable and mask. See details in **Values**, **Variables**, and **Masks** below. ACSC_AMF_EXT_LOOP: Use external loops at corners. The switch requires additional parameters that specify the external loop type, the minimum segment length, and the maximum allowed deviation from profile. ACSC_AMF_LCI_STATE: The switch requires additional parameter that specify LCI state.
Axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to axis 0, ACSC_AXIS_1 to axis 1, etc. After the last axis, one additional element must be located that contains –1 which marks the end of the array. For the axis constants see Axis Definitions.
Point	Array of the final point coordinates. The number and order of values must correspond to the **Axes** array. The **Point** must specify a value for each element of **Axes** except the last element, containing –1 .

Velocity	If ACSC_AMF_VELOCITY flag has been specified, this argument specifies a motion velocity for current segment. Set this argument to ACSC_NONE if not used.
EndVelocity	If ACSC_AMF_ENDVELOCITY flag has been specified, this argument defines the required velocity at the end of the current segment. Set this argument to ACSC_NONE if not used.
Time	If ACSC_AMF_VARTIME flag has been specified, this argument defines the segment processing time in milliseconds, for the current segment only and has no effect on subsequent segments. Set this argument to ACSC_NONE if not used.
Values	Pointer to the string that contains the name of a one-dimensional user-defined array of integer or real type with a size of 10 elements maximum. If ACSC_AMF_USERVARIABLES flag has been specified, this argument defines the values to be written to the **Variables** array at the beginning of the current segment execution. Set this argument to NULL if not used.
Variables	Pointer to the string that contains the name of a one-dimensional user-defined array of the same type and size as **Values** array. If ACSC_AMF_USERVARIABLES flag has been specified, this argument defines the user-defined array, which will be written with **Values** data at the beginning of the current segment execution. Set this argument to NULL if not used.
Index	If ACSC_AMF_USERVARIABLES has not been specified, this argument defines the first element (starting from zero) of the **Variables** array, to which **Values** data will be written to. Set this argument to ACSC_NONE if not used.

Masks	Pointer to the character string that contains the name of a one-dimensional user defined array of integer type and same size as the **Values** array. If ACSC_AMF_USERVARIABLES flag has been specified, this argument defines the masks that are applied to **Values** before the **Values** are written to variables array at the beginning of the current segment execution. The masks are only applied for integer values: *variables(n) = values(n) AND mask(n)* If **Values** is a real array, the masks argument should be NULL. Set this argument to ACSC_NONE if not used.
ExtLoopType	If ACSC_AMF_EXT_LOOP flag has been specified, this argument defines the external loop type. 0 - Cancel external loop 1 – Smooth External loop (line-arc-line) 2 – Triangle External loop (line-line-line) Set this argument to ACSC_NONE if not used.
MinSegmentLength	If ACSC_AMF_EXT_LOOP flag has been specified, this argument defines the Minimum Segment Length. If the lengths of both segments are more than this value, the skywriting algorithm will be applied. Set this argument to ACSC_NONE if not used.
MaxAllowedDeviation	If ACSC_AMF_EXT_LOOP flag has been specified, this argument defines the Maximum Allowed Deviation. The parameter limits the external loop deviation from the defined profile. If the value is negative there is no limitation. Set this argument to ACSC_NONE if not used
LciState	If ACSC_AMF_LCI_STATE flag has been specified, this argument defines the LCI state. LCI_STATE_ON(0) or LCI_STATE_OFF(1) determines the value to be considered the initial state. Set this argument to ACSC_NONE if not used.

Return Value

None if called synchronously, ACSC_WAITBLOCK if called asynchrounously.

Extended error information can be obtained by calling GetErrorString

Comments

The function adds a linear segment that starts at the current point and ends at the destination point to segmented motion.

All axes specified in the Axes array must be specified before the call of the **SegmentLineV2** function. The number and order of the axes in the Axes array must correspond exactly to the number and order of the axes of the **ExtendedSegmentedMotionV2** function.

The Point argument specifies the coordinates of the final point. The coordinates are absolute in the plane.

ACSC_AMF_VELOCITY and ACSC_AMF_VARTIME are mutually exclusive, meaning they cannot be used together.

The function can be called either synchronously or asynchronously, using the relevant function signature.

The controller response indicates that the command was accepted and the segment is added to the motion buffer. The segment can be rejected if the motion buffer is full. In that case, you can call this function periodically until the function returns a non-zero value.

Supported from V3.12.

3.22.3 SegmentLine

Description

The method adds a linear segment to segmented motion that starts at the current point and ends at the destination point.

Syntax

object.SegmentLine(MotionFlags flags, Axis[] axes, double[] point, double velocity, double endVelocity, string values, string variables, int index, string masks)

Async Syntax

ACSC_WAITBLOCK object.SegmentLineAsync(MotionFlags flags, Axis[] axes, double[] point, double velocity, double endVelocity, string values, string variables, int index, string masks)

Arguments

flags	**ACSC_AMF_VELOCITY**: The motion will use the velocity specified for each segment instead of the default velocity. **ACSC_AMF_ENDVELOCITY**: This flag requires an additional parameter that specifies the end velocity. The controller decelerates to the specified velocity at the end of segment. The specified value should be less than the required velocity; otherwise, the parameter is ignored. This flag affects only one segment. This flag also disables corner detection and processing at the end of segment. If this flag is not specified, deceleration is not required. However, in special cases, the deceleration might occur due to corner processing or other velocity control conditions. **ACSC_AMF_USERVARIABLES**: Synchronizes user variables with segment execution. This flag requires additional parameters that specify values, user variable and mask.
axes	Array of axis constants. Each element specifies one involved axis: ACSC_ AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.
point	Array of the coordinates of the initial point on the plane. The number and order of values must correspond to the **axes** array. The **point** must specify a value for each element of **axes** except the last –1 element.
velocity	If the **ACSC_AMF_VELOCITY** flag has been specified, this argument specifies a motion velocity for current segment. Set this argument to **Api.ACSC_NONE** if not used
endVelocity	If the **ACSC_AMF_ENDVELOCITY** flag has been specified, this argument defines required velocity at the end of the current segment. Set this argument to **Api.ACSC_NONE** if not used.
values	Pointer to the character string that contains the name of a one-dimensional user-defined array of integer or real type with a size of 10 elements maximum. If the **ACSC_AMF_USERVARIABLES** flag has been specified, this argument defines the user-defined array, which will be written with values data at the beginning of the current segment execution. Set this argument to **NULL** if not used.

variables	Pointer to the character string that contains the name of a one-dimensional user-defined array of the same type and size as **values** array. If the **ACSC_AMF_USERVARIABLES** flag has been specified, this argument defines the user-defined array, which will be written with values data at the beginning of the current segmentexecution. Set this argument to **NULL** if not used.
index	If the **ACSC_AMF_USERVARIABLES** flag is specified, this argument defines the first element (starting from zero) of variables array, which **Values** data will be written to. Set this argument to **Api.ACSC_NONE** if not used.
masks	Pointerto a bufferthat contains the name of aone-dimensional user defined array of integer type and size as **values** array. If the **ACSC_AMF_USERVARIABLES** flag has been specified, this argument defines the masks that are applied to values before the values are written to the **Variables** array at the beginning of the current segment execution. The masks are only applied for integer values: Variables(n) = values(n) AND mask(n) If **values** is a real array, this argument should be NULL. Set this argument to **NULL** if not used.

Return Value

None

Remarks

The function adds a linear segment that starts in the current point and ends in thedestination point to segmented motion.

All axes specified in the **axes** array must be specified before the call of the *SegmentedMotion* or ExtendedSegmentedMotion methods. The number and order of the axes in the **axes** array must correspond exactly to the number and order of the axes of the **SegmentedMotion** or **ExtendedSegmentedMotion** method.

The **point** argument specifies the coordinates of the final point. The coordinates are absolute in the plane.

The controller response indicates that the command was accepted and the segment is added to the motion buffer. The segment can be rejected if the motion buffer is full. In this case, you can call this function periodically until the function returns a non-zero value.

Example

```
            int timeout = 5000;
             Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
                 Axis.ACSC_NONE };
```

```
        double[] points = { 1000, 1000 };
        Ch.EnableM(axes); // Enable axes 0 and 1
        // Wait axis 0 enabled during 5 sec
        Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
        // Wait axis 1 enabled during 5 sec
        Ch.SegmentedMotion(MotionFlags.ACSC_AMF_VELOCITY, axes,
        points);
        points[0] = -1000;
        points[1] = -1000;
        Ch.SegmentLine(MotionFlags.ACSC_NONE, axes, points,
            Api.ACSC_NONE, Api.ACSC_NONE, null, null,
            Api.ACSC_NONE, null);
        Ch.EndSequenceM(axes);
```

3.22.4 ExtendedSegmentArc1

This function replaces the **SegmentArc1** which is now obsolete.

Description

The method adds an arc segment to a segmented motion and specifies the coordinates of the center point, the coordinates of the final point and the direction of rotation.

Syntax

object.ExtendedSegmentArc1(MotionFlags flags, Axis[] axes, double[] center, double[] finalPoint, RotationDirection rotation, double velocity,
double endVelocity, double time, string values, string variables,
int index, string masks)

Async Syntax

ACSC_WAITBLOCK object.ACSC_WAITBLOCK ExtendedSegmentArc1Async(MotionFlags flags, Axis[] axes, double[] center, double[] finalPoint, RotationDirection rotation,
double velocity, double endVelocity, double time, string values,
string variables, int index, string masks)

Arguments

flags	Bit-mapped parameter that can include one or more of the following flags: **ACSC_AMF_VELOCITY**: The motion will use velocity specified for each segment instead of the default velocity. **ACSC_AMF_ENDVELOCITY**: This flag requires additional parameter that specifies end velocity. The controller decelerates to the specified velocity in the end of segment. The specified value should be less than the required velocity; otherwise the parameter is ignored. This flag affects only one segment. This flag also disables corner detection and processing at the endof segment. If this flag is not specified, deceleration is not required. However, in special case the deceleration might occur due to corner processingor other velocity control conditions. **ACSC_AMF_USERVARIABLES:** Synchronize user variables with segment execution. This flag requires additional parameters that specify values, user variable and mask.
axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.
center	Array of the center coordinates. The number and order of values must correspond to the **axes** array. The **center** must specify a value for each element of the **axes** except the last –1 element.
finalPoint	Array of the final point coordinates. The number and order of values must correspond to the **axes** array. The **finalPoint** must specify a value for each element of **axes** except the last –1 element.
rotation	This parameter defines the direction of rotation. If **rotation** is set to **ACSC_COUNTERCLOCKWISE**, then the rotation is counterclockwise. If **rotation** is set to **ACSC_CLOCKWISE**, then rotation is clockwise.
velocity	If the **ACSC_AMF_VELOCITY** flag has been specified, this argument specifies a motion velocity for current segment. Set this argument to **Api.ACSC_NONE** if not used.
endVelocity	If the **ACSC_AMF_ENDVELOCITY** flag has been specified, this argument defines required velocity at the end of the current segment. Set this argument to **Api.ACSC_NONE** if not used.

time	If ACSC_AMF_VARTIME flag has been specified, this argument defines the segment processing time in milliseconds, for the current segment only and has no effect on subsequent segments. Set this argument to ACSC_NONE if not used.
values	Pointer to the string that contains the name of a one-dimensional user-defined array of integer or real type with a size of 10 elements maximum. If the **ACSC_AMF_USERVARIABLES** flag has been specified, this argument defines the user-defined array, which will be written with values data at the beginning of the current segmentexecution. Set this argument to **NULL** if not used.
variables	Pointer to the string that contains the name of a one-dimensional user-defined array of the same type and size as **Values** array. If the **ACSC_AMF_USERVARIABLES** flag has been specified, this argument defines the user-defined array, which will be written with Values data at the beginning of the current segmentexecution. Set this argument to **NULL** if not used.
index	If the **ACSC_AMF_USERVARIABLES** flag is specified, this argument defines the first element (starting from zero) of variables array, which **values** data will be written to. Set this argument to **Api.ACSC_NONE** if not used.
masks	Pointerto a bufferthat contains the name of aone-dimensional user defined array of integer type and size as **values** array. If the **ACSC_AMF_USERVARIABLES** flag has been specified, this argument defines the masks that are applied to values before the values are written to the **variables** array at the beginning of the current segment execution. The masks are only applied for integer values: Variables(n) = values(n) AND mask(n) If **values** is a real array, this argument should be NULL. Set this argument to **NULL** if not used.

Return Value

None

Remarks

All axes specified in the **axes** array must be specified before the call of the SegmentedMotion ExtendedSegmentedMotion methods. The number and order of the axes in the **Axes** array must correspond exactly to the number and order of the axes of the **SegmentedMotion** or **ExtendedSegmentedMotion** method.

The method waits for the controller response.

The controller response indicates that the command was accepted and the segment is added to the motion buffer. The segment can be rejected if the motion buffer is full. In this case, you can call this method periodically until the method returns non-zero value.

If the method fails, the Error object is filled with the Error Description.

3.22.5 ExtendedSegmentArc2

This function replaces the **SegmentArc2Ext** which is now obsolete.

Description

The method adds an arc segment to a segmented motion and specifies the coordinates of the center point and the rotation angle.

Syntax

object.ExtendedSegmentArc2(MotionFlags flags, Axis[] axes, double[] center, double angle, double[] finalPoint, double velocity, double endVelocity, double time, string values, string variables, int index, string masks)

Async Syntax

object.ACSC_WAITBLOCK ExtendedSegmentArc1Async(MotionFlags flags, Axis[] axes, double[] center, double angle, double[] finalPoint, double velocity, double endVelocity, double time, string values, string variables, int index, string masks)

Arguments

flags	Bit-mapped parameter that can include one or more of the following flags: **ACSC_AMF_VELOCITY**: The motion will use velocity specified for each segment instead of the default velocity. **ACSC_AMF_ENDVELOCITY**: This flag requires additional parameter that specifies end velocity. The controller decelerates to the specified velocity in the end of segment. The specified value should be less than the required velocity; otherwise the parameter is ignored. This flag affects only one segment. This flag also disables corner detection and processing at the endof segment. If this flag is not specified, deceleration is not required. However, in special case the deceleration might occur due to corner processingor other velocity control conditions. **ACSC_AMF_USERVARIABLES**: Synchronize user variables with segment execution. This flag requires additional parameters that specify values, user variable and mask.
axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.
center	Array of the center coordinates. The number and order of values must correspond to the **axes** array. The **center** must specify a value for each element of the **axes** except the last –1 element.
angle	Rotation angle in radians. Positive angle for counterclockwise rotation, negative for clockwise rotation.
finalPoint	Array of the final point coordinates. The number and order of values must correspond to the axes array. The finalPoint must specify a value for each element of axes except the last –1 element.
velocity	If the **ACSC_AMF_VELOCITY** flag has been specified, this argument specifies a motion velocity for current segment. Set this argument to **Api.ACSC_NONE** if not used.
endVelocity	If the **ACSC_AMF_ENDVELOCITY** flag has been specified, this argument defines required velocity at the end of the current segment. Set this argument to **Api.ACSC_NONE** if not used.

time	If ACSC_AMF_VARTIME flag has been specified, this argument defines the segment processing time in milliseconds, for the current segment only and has no effect on subsequent segments. Set this argument to ACSC_NONE if not used.
values	Pointer to the null-terminated character string that contains the name of a one-dimensional user-defined array of integer or real type with a size of 10 elements maximum. If the **ACSC_AMF_USERVARIABLES** flag has been specified, this argument defines the user-defined array, which will be written with values data at the beginning of the current segmentexecution. Set this argument to **NULL** if not used.
variables	Pointer to the string that contains the name of a one-dimensional user-defined array of the same type and size as **values** array. If the **ACSC_AMF_USERVARIABLES** flag has been specified, this argument defines the user-defined array, which will be written with **values** data at the beginning of the current segmentexecution. Set this argument to **NULL** if not used.
index	If the **ACSC_AMF_USERVARIABLES** flag is specified, this argument defines the first element (starting from zero) of variables array, which **values** data will be written to. Set this argument to **Api.ACSC_NONE** if not used.
masks	Pointerto a bufferthat contains the name of aone-dimensional user defined array of integer type and size as **Values** array. If the **ACSC_AMF_USERVARIABLES** flag has been specified, this argument defines the masks that are applied to values before the values are written to the **variables** array at the beginning of the current segment execution. The masks are only applied for integer values: Variables(n) = values(n) AND mask(n) If **values** is a real array, this argument should be NULL. Set this argument to **Api.ACSC_NONE** if not used.

Return Value

None

Remarks

The method adds an arc segment to the segmented motion and specifies the coordinates of the center point and the rotation angle.

All axes specified in the **axes** array must be specified before the call of the *SegmentedMotion ExtendedSegmentedMotion* methods. The number and order of the axes in the **axes** array must correspond exactly to the number and order of the axes of the *SegmentedMotion* or *ExtendedSegmentedMotion* method.

The method waits for the controller response.

The controller response indicates that the command was accepted and the segment is added to the motion buffer. The segment can be rejected if the motion buffer is full. In this case, you can call this method periodically until the method returns a non-zero value.

If the method fails, the Error object is filled with the Error Description.

Example

```
bool success = true;
Axis[] Axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1, Axis.ACSC_AXIS_2,
Axis.ACSC_AXIS_3, Axis.ACSC_NONE };
string res1, res2;

Ch.EnableMAsync(Axes);
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, 5000);
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, 5000);
Ch.SetFPositionAsync(Axes[0], 1000);
Ch.SetFPositionAsync(Axes[1], 1000);
Ch.SetFPositionAsync(Axes[2], 500);
Ch.SetFPositionAsync(Axes[3], 500);

double[] Point1 = { 1000, 1000, 500, 500 };
Ch.ExtendedSegmentedMotion(MotionFlags.ACSC_NONE, Axes, Point1,
Api.ACSC_NONE, Api.ACSC_NONE, Api.ACSC_NONE, Api.ACSC_NONE,
Api.ACSC_NONE, null);

double[] center = { 1000, 0 };
double[] FinalPoint = { 1000, -1000, 500, -500 };
Ch.ExtendedSegmentArc1(MotionFlags.ACSC_NONE, Axes, center, FinalPoint,
RotationDirection.ACSC_CLOCKWISE, Api.ACSC_NONE, Api.ACSC_NONE,
Api.ACSC_NONE, null, null, Api.ACSC_NONE, null);

Point1[0] = -1000; Point1[1] = -1000; Point1[2] = -500; Point1[3] = -500;

Ch.Line(Axes, Point1);

center[0] = -1000; center[1] = 0;
double[] FinalPoint2 = { -500, 500 };
Ch.ExtendedSegmentArc2(MotionFlags.ACSC_NONE, Axes, center, -3.14159,
FinalPoint2, Api.ACSC_NONE, Api.ACSC_NONE, Api.ACSC_NONE, null, null,
Api.ACSC_NONE, null);

Point1[0] = 1000; Point1[1] = 1000; Point1[2] = 500; Point1[3] = 500;

Ch.Line(Axes, Point1);

Ch.EndSequenceMAsync(Axes);
```

```
Ch.WaitMotionEnd(Axis.ACSC_AXIS_0, 20000);
Ch.WaitMotionEnd(Axis.ACSC_AXIS_1, 20000);
Ch.WaitMotionEnd(Axis.ACSC_AXIS_2, 20000);
Ch.WaitMotionEnd(Axis.ACSC_AXIS_3, 20000);
```

3.22.6 SegmentArc2V2

Description

The function adds an arc segment to a segmented motion and specifies the coordinates of the center point and the rotation angle.

Syntax

public void SegmentArc2V2(MotionFlags flags, Axis[] axes, double[] center, double angle, double[] finalPoint, double velocity, double endVelocity, double time, string values, string variables, int index, string masks, int extLoopType, double minSegmentLength, double maxAllowedDeviation, int lciState)

Arguments

Flags	Bit-mapped argument that can include one or more of the following flags: ACSC_AMF_VELOCITY: the motion will use velocity specified for each segment instead of the default velocity. ACSC_AMF_ENDVELOCITY: this flag requires an additional parameter that specifies end velocity. The controller decelerates to the specified velocity in the end of segment. The specified value should be less than the required velocity; otherwise the argument is ignored. This flag affects only one segment. ACSC_AMF_VARTIME: this flag requires an addition parameter that specifies the segment processing time in milliseconds. The segment processing time defines velocity at the current segment only and has no effect on subsequent segments. This flag also disables corner detection and processing at the end of segment. If this flag is not specified, deceleration is not required. However, in special cases the deceleration might occur due to corner processing or other velocity control conditions. ACSC_AMF_USERVARIABLES: synchronize user variables with segment execution. This flag requires additional parameters that specify values, user variable and mask. See details in the **Values**, **Variables**, and **Masks** arguments below. ACSC_AMF_EXT_LOOP: Use external loops at corners. The switch requires additional parameters that specify the external loop type, the minimum segment length, and the maximum allowed deviation from profile. ACSC_AMF_LCI_STATE: The switch requires an additional parameter that specifies LCI state.
Axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to 0, ACSC_AXIS_1 to 1, etc. After the last axis, one additional element must be located that contains –1 which marks the end of the array. For the axis constants see Axis Definitions.
Center	Array of the center coordinates. The number and order of values must correspond to the **Axes** array. The **Center** must specify a value for each element of the **Axes** except the last–1 element.

Angle	Rotation angle in radians. Positive angle for counterclockwise rotation, negative for clockwise rotation.
FinalPoints	Array indicating the final points of the secondary axes, array size must be number of secondary axes (size of **Axes** – 2). Set this argument to **NULL** if not used.
Velocity	If ACSC_AMF_VELOCITY flag has been specified, this argument specifies a motion velocity for current segment. Set this argument to ACSC_NONE if not used.
EndVelocity	If ACSC_AMF_ENDVELOCITY flag has been specified, this argument defines required velocity at the end of the current segment. Set this argument to ACSC_NONE if not used.
Time	If ACSC_AMF_VARTIME flag has been specified, this argument defines the segment processing time in milliseconds, for the current segment only and has no effect on subsequent segments. Set this argument to ACSC_NONE if not used.
Values	Pointer to the string that contains the name of a one-dimensional user-defined array of integer or real type with a size of 10 elements maximum. If ACSC_AMF_USERVARIABLES flag has been specified, this argument defines the values to be written to the **Variables** array at the beginning of the current segment execution. Set this argument to NULL if not used.
Variables	Pointer to string that contains the name of a one-dimensional user-defined array of the same type and size as **Values** array. If ACSC_AMF_USERVARIABLES flag has been specified, this argument defines the user-defined array, which will be written with **Values** data at the beginning of the current segment execution. Set this argument to NULL if not used.
Index	If ACSC_AMF_USERVARIABLES has not been specified, this argument defines the first element (starting from zero) of the **Variables** array, to which **Values** data will be written to. Set this argument to ACSC_NONE if not used.

Masks	Pointer to the string that contains the name of a one-dimensional user defined array of integer type and same size as the **Values** array. If ACSC_AMF_USERVARIABLES flag has been specified, this argument defines the masks that are applied to **Values** before the **Values** are written to variables array at the beginning of the current segment execution. The masks are only applied for integer values: *variables(n) = values(n) AND mask(n)* If **Values** is a real array, the masks argument should be NULL. Set this argument to ACSC_NONE if not used.
ExtLoopType	If ACSC_AMF_EXT_LOOP flag has been specified, this argument defines the external loop type. 0 - Cancel external loop 1 – Smooth External loop (line-arc-line) 2 – Triangle External loop (line-line-line) Set this argument to ACSC_NONE if not used
MinSegmentLength	If ACSC_AMF_EXT_LOOP flag has been specified, this argument defines the Minimum Segment Length.If the lengths of both segments are more than this value, the skywriting algorithm will be applied. Set this argument to ACSC_NONE if not used.
MaxAllowedDeviation	If ACSC_AMF_EXT_LOOP flag has been specified, this argument defines the Maximum Allowed Deviation. The parameter limits the external loop deviation from the defined profile. If the value is negative there is no limitation. Set this argument to ACSC_NONE if not used.
LciState	If ACSC_AMF_LCI_STATE flag has been specified, this argument defines the LCI state. LCI_STATE_ON(0) or LCI_STATE_OFF(1) determines the value to be considered the initial state. Set this argument to ACSC_NONE if not used.

Return Value

None if called synchronously, ACSC_WAITBLOCK if called asynchrounously.

Extended error information can be obtained by calling GetErrorString

Comments

All axes specified in the **Axes** array must be specified before calling the function. The number and order of the axes in the **Axes** array must correspond exactly to the number and order of the axes of the ExtendedSegmentedMotionV2 function.

The **Point** argument specifies the coordinates of the final point. The coordinates are absolute in the plane.

ACSC_AMF_VELOCITY and ACSC_AMF_VARTIME are mutually exclusive, meaning they cannot be used together.

The controller response indicates that the command was accepted and the segment is added to the motion buffer. The segment can be rejected if the motion buffer is full. In that case, you can call this function periodically until the function returns a non-zero value.

Supported from V3.12.

3.22.7 Stopper

Description

The method provides a smooth transition between two segments of segmentedmotion.

Syntax

object.Stopper(Axis[] axes)

Async Syntax

ACSC_WAITBLOCK object.StopperAsync(Axis[] axes)

Arguments

axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.

Return Value

None

Remarks

The controller builds the motion so that the vector velocity follows a smooth velocity diagram. The segments define the projection of the vector velocity to axis velocities. If all segments are connected smoothly, axis velocity is also smooth. However, if the user defined a path withan inflection point, axis velocity has a jump in this point. The jump can cause a motion failure due to the acceleration limit.

The method is used to avoid velocity jump in the inflection points. If the method is specified between two segments, the controller provides smooth deceleration to zero in the end of first segment and smooth acceleration to specified velocity in the beginning of second segment.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
    Axis.ACSC_NONE };
double[] points = { 1000, 1000 };
// Create segmented motion,coordinates of the initial point
// are(1000, 1000)
Ch.EnableM(axes);
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Wait axis 1 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
Ch.Segment(MotionFlags.ACSC_NONE, axes, points);
// Add line segment with final point (1000,-1000),vector
// velocity 25000
points[0] = 1000;
points[1] = -1000;
Ch.ExtLine(axes, points, 25000);
Ch.Stopper(axes);
// Add line segment with final point (-1000,-1000),vector
// velocity 25000
points[0] = -1000;
points[1] = -1000;
Ch.ExtLine(axes, points, 25000);
Ch.Stopper(axes);
// Add line segment with final point (-1000,1000),vector
// velocity 25000
points[0] = -1000;
points[1] = 1000;
Ch.ExtLine(axes, points, 25000);
Ch.Stopper(axes);
// Add line segment with final point (1000,1000), vector
// velocity 25000
points[0] = 1000;
points[1] = 1000;
Ch.ExtLine(axes, points, 25000);
// Finish the motion
// End of the multi-point motion
Ch.EndSequenceM(axes);
```

3.22.8 Projection

Description

The method sets a projection matrix for segmented motion.

Syntax

object.Projection(Axis[] axes, stringmatrix)

Async Syntax

ACSC_WAITBLOCK object.ProjectionAsync(Axis[] axes, stringmatrix)

Arguments

axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.
matrix	Pointer to the string containing the name of the matrix that provides the specified projection.

Return Value

None

Remarks

The method sets a projection matrix for segmented motion.

The projection matrix connects the plane coordinates and the axis values in the axis group. The projection can provide any transformation as rotation or scaling. The number of the matrix rows must be equal to the number of the specified axes. The number of the matrix columns must equal two.

The matrix must be declared before as a global variable by an ACSPL+ program or by the

DeclareVariable method and must be initialized by an ACSPL+ program or by the method. For more information about projection, see the *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1, Axis.ACSC_NONE };
double[] points = { 10000, 10000 }; double[] center = { 1000, 0 }; double
[,] matrix = new double[3, 2]; matrix[0, 0] = 1; matrix[0, 1] = 0; matrix
[1, 0] = 0; matrix[1, 1] = 1.41421; matrix[2, 0] = 0; matrix[2, 1] =
1.41421;
// Declare the matrix that will contain the projection
api.DeclareVariable(AcsplVariableType.ACSC_REAL_TYPE, "ProjectionMatrix
(3), (2)");
// Initialize the projection matrix
Ch.WriteVariable(matrix,”ProjectionMatrix”,ProgramBuffer.ACSC_NONE);
Ch.EnableM(axes);
// Create a group of the involved axes
api.Group(axes);
// Create segmented motion, coordinates of the initial point are
(10000,10000)
api.Segment(MotionFlags.ACSC_NONE, axes, points);
// Incline the working plance XY by 45 degrees.
```

```
axes = new Axis[] { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1, Axis.ACSC_AXIS_1,
Axis.ACSC_NONE };
api.Projection(axes, "ProjectionMatrix");
// Describe circle with center (1000, 0) clockwise rotation.
// Although the circle was defined,really on the plane XY we will get
// the Ellipse stretched along the Y axis
axes = new Axis[] { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1, Axis.ACSC_NONE };
api.Arc2(axes, center, -2 * 3.14159);
api.EndSequenceM(axes);
```

3.23 Blended Segmented Motion Functions

The Blended Segmented Motion Functions are:

Function	Description
BlendedSegmentMotion	The function initiates a multi-axis blended segmented motion
BlendedLine	The function adds a linear segment that starts at the current point and ends at the destination point of segmented motion.
BlendedArc1	The function adds to the motion path an arc segment that starts at the current point and ends at the destination point with the specified center point.
BlendedArc2	The function adds an arc segment to a segmented motion and specifies the coordinates of the center point and the rotation angle.

3.23.1 BlendedSegmentMotion

Description

The function initiates a multi-axis blended segmented motion.

Syntax

object.BlendedSegmentMotion(MotionFlags flags, Axis[] axes, , double[] Position, double SegmentTime, double AccelerationTime, double JerkTime, double DwellTime, ACSC_WAITBLOCK wait)

Arguments

flags	Bit-mapped parameter that can include one or more of the following flags: **ACSC_AMF_WAIT**: plan the motion but don't start it until the method GoM is executed. **ACSC_AMF_VELOCITY**: the motion will use velocity specified for each segment instead of the default velocity. **ACSC_AMF_CYCLIC**: the motion uses the segment sequence as a cyclic array: after the last segment, move along the first segment etc. **ACSC_AMF_VELOCITYLOCK**: slaved motion: the motion advances in accordance to the master value of the leading axis. **ACSC_AMF_POSITIONLOCK**: slaved motion, strictly conformed to master. **ACSC_AMF_EXTRAPOLATED**: if a master value travels beyond the specified path, the last or the first segment is extrapolated. **ACSC_AMF_STALLED**: if a master value travels beyond the specified path, the motion stalls at the last or first point.
axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.
Position	Array of the starting coordinates. The number and order of values must correspond to the Axes array. The Center must specify a value for each element of the Axes except the last –1 element.
SegmentTime	This parameter will set the default initial segment time in milliseconds.
AccelerationTime	This parameter will set the default Acceleration time in milliseconds.
JerkTime	This parameter will set the default Jerk time in milliseconds.
DwellTime	If ACSC_AMF_DWELLTIME is set, this parameter will set the initial dwell time between segments in milliseconds. If this argument is specified, no blending will be done for all segments of the motion. That means that the motion will be stopped at the end of each segment for the specified DwellTime milliseconds.
wait	Wait block returned by Async method.

Return Value

None

Remarks

Blended segmented motion is a type of segmented motion that doesn’t provide look-ahead capabilities, unlike Extended segmented motion. Both type of motions are intended for processing a complex multi-axis trajectory and smoothing corners between segments, but do it in different ways. The Extended segmented motion (**XSEG**) allows achieving highest throughput within the defined axis limitations and the defined accuracy. The blended segmented motion (**BSEG**) allows passing along the trajectory with the defined timing constrains. The function itself does not specify any movement, so the created motion starts only after the first segment is specified.

The segments of motion are specified by using BlendedLine, BlendedArc1, BlendedArc2functions that follow this function.

The motion finishes when the **EndSequenceM** function is executed. If the call to **EndSequenceM** is omitted, the motion will stop at the last segment of the sequence and wait for the next segment. No transition to the next motion in the motion queue will occur until the function **EndSequenceM** is executed.

The function can wait for the controller response or can return immediately as specified by the Wait argument. The controller response indicates that the command was accepted and the motion was planned successfully. The function does not wait and does not validate the end of the motion. To wait for the motion end, use the acsc_WaitMotionEnd function.

If Wait points to a valid ACSC_WAITBLOCK structure, the calling thread must not use or delete the Wait item until a call to the acsc_WaitForAsyncCall function.

Example

```
Axis Axis0 = Axis.ACSC_AXIS_0;
Axis Axis1= Axis.ACSC_AXIS_1;
Axis[] Axes = { Axis0, Axis1, Axis.ACSC_NONE };
Api.transaction($“enable {Axis0}”);
Api.transaction($“commut {Axis0}”);
Api.transaction($“enable {Axis1}”);
Api.transaction($“commut {Axis1}”);

double[] Point = { 1000, 1000 };
api.BlendedSegmentMotion(MotionFlags.ACSC_NONE, Axes,
Point,//Starting point of motion
1000, // Segment time
500, // Segment Acceleration time
200, // Segment jerk time
0 //Segment Dwell time
);
double[] Center = { 1000, 0 };
double[] FinalPoint = { 1000, -1000 };
api.BlendedArc1(MotionFlags.ACSC_NONE, Axes, Center, FinalPoint,
RotationDirection.ACSC_CLOCKWISE, 1000, 200, 300, 0);
FinalPoint[0] = -1000; FinalPoint[1] = -1000; ;
```

```
api.BlendedLine(MotionFlags.ACSC_NONE, Axes, FinalPoint, 1000,
200, 300, 0);
Center[0] = -1000; Center[1] = 0;
double[] FinalPoint2 = { -500, 500 };
api.BlendedArc2(MotionFlags.ACSC_NONE, Axes, Center,
-3.141529, //Angle
1000, 200, 300, 100);
FinalPoint[0] = 1000; FinalPoint[1] = 1000;
api.BlendedLine(MotionFlags.ACSC_NONE, Axes, FinalPoint, 1000,
200, 300, 0);
api.EndSequenceM(Axes);
```

3.23.2 BlendedLine

Description

The function adds a linear segment that starts at the current point and ends at the destination point of segmented motion.

Syntax

int object.BlendedLine(MotionFlags flags, Axis[] axes, double[] point,double SegmentTime,double AccelerationTime, double JerkTime, double DwellTime

Async Syntax

ACSC_WAITBLOCK object.BlendedLine(MotionFlags flags, Axis[] axes, double[] point, double SegmentTime,double AccelerationTime, double JerkTime, double DwellTime)

Arguments

flags	Bit-mapped argument that can include one or more of the following flags: ACSC_AMF_BSEGTIME: This flag requires an additional parameter that defines the required segment time in milliseconds. ACSC_AMF_BSEGACC: This flag requires an additional parameter that defines the required segment acceleration time in milliseconds. ACSC_AMF_BSEGJERK: This flag requires an additional parameter that defines the required jerk time in milliseconds. ACSC_AMF_DWELLTIME: This flag requires an additional parameter that specifies the dwell time, in milliseconds, at the final point of the segment.
axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.

Point	Array of the final point coordinates. The number and order of values must correspond to the **Axes** array. The **Point** must specify a value for each element of Axes except the last –1 element.
SegmentTime	If ACSC_AMF_BSEGTIME is set, this parameter will set the segment time, in milliseconds, for the current and all following segments – until the parameter is redefined.
AccelerationTime	If ACSC_AMF_BSEGACC is set, this parameter will set the Acceleration time, in milliseconds, for the current and all following segments – until the parameter is redefined.
JerkTime	If ACSC_AMF_BSEGJERK is set, this parameter will set the default Jerk time, in milliseconds, for the current and all following segments – until the parameter is redefined.
DwellTime	If ACSC_AMF_DWELLTIME is set, this parameter will set the initial dwell time between segments in milliseconds. If this argument is specified, no blending will be done for all segments of the motion. That means that the motion will be stopped at the end of each segment for the specified DwellTime milliseconds.

Return Value

None from synchronous version, ACSC_WAITBLOCK from async version.

Remarks

The function adds a linear segment that starts at the current point and ends at the destination point to segmented motion.

All axes specified in the **Axes** array must be specified in a previous call to the BlendedSegmentMotionfunction. The number and order of the axes in the **Axes** array must correspond exactly to the number and order of the axes of the call to the **BlendedSegmentMotion** function.

The **Point** argument specifies the coordinates of the final point. The coordinates are absolute in the plane.

The function can wait for the controller response or can return immediately as specified by the Wait argument.

The controller response indicates that the command was accepted and the segment is added to the motion buffer. The segment can be rejected if the motion buffer is full. In that case, you can call this function periodically until the function returns a non-zero value.

Example

```
Axis[] Axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1, Axis.ACSC_NONE };
double[] Point = { 1000, 1000 };
api.BlendedSegmentMotion(MotionFlags.ACSC_NONE, Axes, Point,//Starting
point of motion
```

```
1000, // Segment time
500,  // Segment Acceleration time
200,  // Segment jerk time
0     // Segment Dwell time
);
double[] Center = { 1000, 0 };
double[] FinalPoint = { -1000, -1000 };
api.BlendedLine(MotionFlags.ACSC_NONE, Axes,
FinalPoint, 1000, 200, 300, 0);
api.EndSequenceM(Axes);
```

3.23.3 BlendedArc1

Description

The function adds to the motion path an arc segment that starts at the current point and ends at the destination point with the specified center point.

Syntax

Int object.BlendedArc1(MotionFlags flags, Axis[] axes, double[] center, double[] FinalPoint, int Rotation, double SegmentTime, double JerkTime, double DwellTime);

Async Syntax

ACSC_WAITBLOCK object.AsyncBlendedArc1(MotionFlags flags, Axis[] axes, double[] center, double[] FinalPoint, int Rotation, double SegmentTime, double JerkTime, double DwellTime);

Arguments

flags	Bit-mapped argument that can include one or more of the following flags: ACSC_AMF_BSEGTIME: This flag requires an additional parameter that defines the required segment time in milliseconds. ACSC_AMF_BSEGACC: This flag requires an additional parameter that defines the required segment acceleration time in milliseconds. ACSC_AMF_BSEGJERK: This flag requires an additional parameter that defines the required jerk time in milliseconds. ACSC_AMF_DWELLTIME: This flag requires an additional parameter that specifies the dwell time, in milliseconds, at the final point of the segment.
axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.

Center	Array of the center coordinates. The number and order of values must correspond to the **Axes** array. The **Center** must specify a value for each element of the **Axes** except the last–1 element.
FinalPoint	Array of the final point coordinates. The number and order of values must correspond to the **Axes** array. The **FinalPoint** must specify a value for each element of **Axes** except the last –1 element.
Rotation	This argument defines the direction of rotation. If **Rotation** is set to **ACSC_ COUNTERCLOCKWISE**, then the rotation is counterclockwise. If **Rotation** is set to **ACSC_CLOCKWISE**, then rotation is clockwise.
SegmentTime	If **ACSC_AMF_BSEGTIME** is set, this parameter will set the segment time, in milliseconds, for the current and all following segments – until the parameter is redefined.
AccelerationTime	If **ACSC_AMF_BSEGACC** is set, this parameter will set the Acceleration time, in milliseconds, for the current and all following segments – until the parameter is redefined.
JerkTime	If **ACSC_AMF_BSEGJERK** is set, this parameter will set the default Jerk time, in milliseconds, for the current and all following segments – until the parameter is redefined.
DwellTime	If **ACSC_AMF_DWELLTIME** is set, this parameter will set the dwell time between segments in milliseconds. If this argument is specified, no blending will be done for all segments of the motion. That means that the motion will be stopped at the end of each segment for the specified DwellTime milliseconds.

Return Value

None from synchronous version, ACSC_WAITBLOCK from async version.

Remarks

All axes specified in the Axes array must be specified before the call of the BlendedSegmentMotionfunction. The number and order of the axes in the **Axes** array must correspond exactly to the number and order of the axes of the **BlendedSegmentMotion** function.

The **Point** argument specifies the coordinates of the final point. The coordinates are absolute in the plane.

ACSC_AMF_VELOCITY and **ACSC_AMF_VARTIME** are mutually exclusive, meaning they cannot be used together.

The function can wait for the controller response or can return immediately as specified by the **WAIT** argument. The controller response indicates that the command was accepted and the segment is added to the motion buffer. The segment can be rejected if the motion buffer is full. In that case, you can call this function periodically until the function returns a non-zero value.

Example

```
Axis[] Axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1, Axis.ACSC_NONE };
double[] Point = { 1000, 1000 };

api.BlendedSegmentMotion(MotionFlags.ACSC_NONE, Axes,
Point,// Starting point of motion
1000, // Segment time
500,  // Segment Acceleration time
200,  // Segment jerk time
0     // Segment Dwell time
);
double[] Center = { 1000, 0 };
double[] FinalPoint = { 1000, -1000 };
api.BlendedArc1(MotionFlags.ACSC_NONE, Axes, Center,
FinalPoint, RotationDirection.ACSC_CLOCKWISE,
1000, // Segment time
200,  // Segment Acceleration time
300,  // Segment jerk time
0     // Segment Dwell time
);api.EndSequenceM(Axes);
```

3.23.4 BlendedArc2

Description

The function adds an arc segment to a segmented motion and specifies the coordinates of the center point and the rotation angle.

Syntax

Int BlendedArc2 (int Flags, int* Axes, double* Center, double Angle, double SegmentTime, double AccelerationTime, double JerkTime, double DwellTime);

Async Syntax

ACSC_WAITBLOCK acsc_BlendedArc2 (int Flags, int* Axes, double* Center, double Angle, double SegmentTime, double AccelerationTime, double JerkTime, double DwellTime);

Arguments

flags	Bit-mapped argument that can include one or more of the following flags: **ACSC_AMF_BSEGTIME**: This flag requires an additional parameter that defines the required segment time in milliseconds. **ACSC_AMF_BSEGACC**: This flag requires an additional parameter that defines the required segment acceleration time in milliseconds. **ACSC_AMF_BSEGJERK**: This flag requires an additional parameter that defines the required jerk time in milliseconds. **ACSC_AMF_DWELLTIME**: This flag requires an additional parameter that specifies the dwell time, in milliseconds, at the final point of the segment.
axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.
Center	Array of the center coordinates. The number and order of values must correspond to the **Axes** array. The **Center** must specify a value for each element of the **Axes** except the last–1 element.
Angle	Rotation angle in radians. Positive angle for counterclockwise rotation, negative for clockwise rotation.
SegmentTime	If **ACSC_AMF_BSEGTIME** is set, this parameter will set the segment time, in milliseconds, for the current and all following segments – until the parameter is redefined.
AccelerationTime	If **ACSC_AMF_BSEGACC** is set, this parameter will set the acceleration time, in milliseconds, for the current and all following segments – until the parameter is redefined.
JerkTime	If **ACSC_AMF_BSEGJERK** is set, this parameter will set the default Jerk time, in milliseconds, for the current and all following segments – until the parameter is redefined.
DwellTime	If **ACSC_AMF_DWELLTIME** is set, this parameter will set the dwell time between segments in milliseconds. If this argument is specified, no blending will be done for all segments of the motion. That means that the motion will be stopped at the end of each segment for the specified DwellTime milliseconds.

Return Value

None from synchronous version, ACSC_WAITBLOCK from async version.

Remarks

All axes specified in the Axes array must be specified before the call of the BlendedSegmentMotionfunction. The number and order of the axes in the **Axes** array must correspond exactly to the number and order of the axes of the **BlendedSegmentMotion** function.

The **Center** argument specifies the coordinates of the center point. The coordinates are absolute in the plane.

The function can wait for the controller response or can return immediately as specified by the **WAIT** argument. The controller response indicates that the command was accepted and the segment is added to the motion buffer. The segment can be rejected if the motion buffer is full. In that case, you can call this function periodically until the function returns a non-zero value.

Example

```
Axis[] Axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1, Axis.ACSC_NONE };
double[] Point = { 1000, 1000 };
api.BlendedSegmentMotion(MotionFlags.ACSC_NONE, Axes,
Point,//Starting point of motion
1000, // Segment time
500,  // Segment Acceleration time
200,  // Segment jerk time
0     // Segment Dwell time
);
double[] Center = { 1000, 0 };
api.BlendedArc2(MotionFlags.ACSC_NONE, Axes, Center,
-3.14159, // Angle
1000, 200, 300, 100);
api.EndSequenceM(Axes);
```

3.24 NURBS and SPATH Methods

3.24.1 NurbsPoint

Description

The function adds a new control point to the NURBS motion generator.

Syntax

void NurbsPoint(MotionFlags Flags , Axis [] Axes, double [] Point, double Velocity, double Required_vel, double Knot, double Weight);

Arguments

Flags	Bit-mapped argument that may include one or more of the following flags: **ACSC_AMF_VELOCITY** - The motion will use the velocity specified for each segment instead of the default velocity. **ACSC_AMF_REQUIRED_VELOCITY** - Specify required velocity. This flag is not compatible with **ACSC_AMF_VELOCITY**. The flag requires a non-zero value in the parameter that specifies required velocity. The value is considered required velocity at the current point, but does not change required velocity for subsequent points. **ACSC_AMF_CORNER** - Mark the current point as a corner. The control point is processed as a corner. Actually, such point divides the spline into two independent splines. **ACSC_AMF_DUMMY** - Mark the point specification as dummy. Dummy point specifications can either precede the first control point specification, or follow the last control point specification. Dummy points specification is required in rare cases where default calculation of starting/trailing knots is not suitable (see Dummy point specification) **ACSC_AMF_WEIGHT** - Specify control point weight. The suffix requires additional parameter that specifies the weight of the control point. **ACSC_AMF_KNOT** - Specify knot delta. The suffix requires additional parameter that specifies knot delta from the previous knot. The new knot is calculated as the previous knot plus delta.
Axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to axis 0, ACSC_AXIS_1 to axis 1, etc. After the last axis, one additional element must be included that contains –1, marking the end of the array. For the axis constants see Axis Definitions.
Point	Array of the final point coordinates. The number and order of values must correspond to the Axes array. The Point must specify a value for each element of Axes except the last –1 element.
Velocity	If the **ACSC_AMF_VELOCITY** flag was specified, this argument specifies a motion velocity for current segment. Set this argument to **ACSC_NONE** if not used.
Required_vel	If the **ACSC_AMF_REQUIRED_VELOCITY** flag was specified, this argument specifies a Required velocity in the current point. Set this argument to **ACSC_NONE** if not used.
Knot	If **ACSC_AMF_ KNOT** flag was specified, this argument specifies a knot delta from the previous knot. Set this argument to **ACSC_NONE** if not used.

Weight	If **ACSC_AMF_ WEIGHT** flag has been specified, this argument defines the weight of the control point. Set this argument to **ACSC_NONE** if not used

Return Value

None

Comments

WARNING KNOTS and WEIGHTS parameters should be properly calculated for NURBS. Using random values can cause unexpected behavior of the NURBS algorithm.

Example

```
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1, Axis.ACSC_NONE };
               double[] nPoint = { 0.0,0.0 };
               double[][] pointsArray = new double[][] {
                       new double[] { 4.0, -6.0 },
                       new double[] { -4.0, 1.0 },
                       new double[] { -1.5, 5.0 },
                       new double[] { 0.0, 2.0 },
                       new double[] { 1.5, 5.0 },
                       new double[] { 4.0, 1.0 },
                       new double[] { -4.0, -6.0 }
               };

            // Enable motors before using the Nurbs motion.
            acsApi.Enable(Axis.ACSC_AXIS_0);
            acsApi.WaitMotorEnabled(Axis.ACSC_AXIS_0,1,5000);

            acsApi.Enable(Axis.ACSC_AXIS_1);
            acsApi.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, 5000);

            acsApi.NurbsMotion(
                MotionFlags.ACSC_AMF_VELOCITY, // Specified Velocity
flag.
                axes,    // Array of axis
                100,     // Velocity
                -1,      // Exception Angle
                -1,      // Exception Length
                -1,      // Motion Delay
                null     // Segments
                );

           // Setting the nurbs points
            for (int i = 0; i < 7; i++)
            {
                nPoint[0] = pointsArray[i][0];
```

```
            nPoint[1] = pointsArray[i][1];
            acsApi.NurbsPoints(
                MotionFlags.ACSC_NONE,      // Motion Flags
                axes,        // Array of axis
                nPoint,      // Segment Point
                -1,          //  Velocity
                -1,          // Required Velocity
                -1,          // Knot
                -1           // Weight
                );
        }

        acsApi.EndSequenceM(axes);
```

3.24.2 NurbsMotion

Description

The function creates NURBS motion. NURBS is a motion generator based on the NURBS spline specification. The algorithm accepts as parameters points, weights, and knots, and generates a spline trajectory accordingly.

Syntax

void NurbsMotion(MotionFlags Flags, Axis [] Axes , double Velocity, double ExceptionAngle, double ExceptionLength, double MotionDelay, string Segments);

Arguments

Flags	Bit-mapped argument that can include one or more of the following flags: **ACSC_AMF_WAIT** - plan the motion but do not start it until the function **acsc_GoM** is executed. **ACSC_AMF_VELOCITY** - the motion will use the velocity specified for each segment instead of the default velocity. **ACSC_AMF_EXT_DELAY_MOTION** - defines actual motor movement delay in microseconds. The delay resolution is 50 microseconds. The flag requires an additional parameter that specifies the motion delay. **ACSC_AMF_NURBS_CONSIDER_ACC** - Acceleration consideration. Allow the MG to deviate from specified axes acceleration parameter during velocity profile generation. **ACSC_AMF_NURBS_EXCEPTION_ANGLE** - The flag requires an additional parameter that specifies maximum angle in a control point. The value defines exceptions from spline interpolation. If for an internal control point, directions to the previous and the next control points require direction change more than the specified angle (by modulo), the control point is processed as a corner. Actually, defining such a point divides the spline into two independent splines. **ACSC_AMF_NURBS_EXCEPTION_LENGTH** - Specify exception length. This flag requires an additional parameter that specifies maximum segment length. The value defines exceptions from spline interpolation. If a distance between two control points appears longer than the specified length, the trajectory between the points is considered straight. Actually, two independent splines are built before and after the segment.
Axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to axis 0, ACSC_AXIS_1 to axis 1, etc. After the last axis, one additional element must be located that contains –1 which marks the end of the array. For the axis constants see Axis Definitions.
Velocity	If the **ACSC_AMF_VELOCITY** flag was specified, this argument specifies a motion velocity for current segment. Set this argument to **ACSC_NONE** if not used.
ExceptionAngle	If the **ACSC_AMF_NURBS_EXCEPTION_ANGLE** flag was specified, this argument defines ExceptionAngle. Set this argument to **ACSC_NONE** if not used.

ExceptionLength	If ACSC_AMF_ NURBS_EXCEPTION_LENGTH flag was specified, this argument defines ExceptionLength. Set this argument to ACSC_NONE if not used.
MotionDelay	If ACSC_AMF_DELAY_MOTION flag has been specified, this argument defines the bit motion delay. Defines actual motor movement delay in microseconds. The delay resolution is 50 microseconds. The maximum delay is 100 controller cycles or 100ms for CTIME=1ms or 20ms for CTIME=0.2ms. Set this argument to ACSC_NONE if not used
Segments	String that contains the name of a one-dimensional user-defined array used to store added segments. Set this argument to NULL if not used. By default, if this argument is not specified, the controller allocates internal buffer for storing 50 segments only. The argument allows the user application to reallocate the buffer for storing a larger number of segments. The larger number of segments may be required if the application needs to add many very small segments in advanced. For most applications, the internal buffer size is enough and should not be enlarged. The buffer is for the controller internal use only and should not be used by the user application. The buffer size calculation rule: each segment requires about 600 bytes, so if it is necessary to allocate the buffer for 200 segments, it should be at least 600 * 200 = 120,000 bytes. The following declaration defines a 120,000 bytes buffer: real buf(15000) See XARRSIZE explanation in the ACSPL+ Command and Variable Reference Guide for details on how to declare a buffer with more than 100,000 elements.

Return Value

None

Example

```
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1, Axis.ACSC_NONE };

// Enable motors before using the Nurbs motion.
acsApi.Enable(Axis.ACSC_AXIS_0);
acsApi.WaitMotorEnabled(Axis.ACSC_AXIS_0,1,5000);

  acsApi.Enable(Axis.ACSC_AXIS_1);
  acsApi.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, 5000);
```

```
        acsApi.NurbsMotion(
         MotionFlags.ACSC_AMF_VELOCITY, // Specified Velocity flag.
                  axes,    // Array of axis
                  100,     // Velocity
                  -1,      // Exception Angle
                  -1,      // Exception Length
                  -1,      // Motion Delay
                  null     // Segments
                  );

         acsApi.EndSequenceM(axes);
```

3.24.3 SmoothPathMotion

Description

Smooth path is a motion type that accepts line segments and generates a spline motion between the specified points.

The motion deviates from exact coordinates after interpolation.

To control the deviation, the user can generate more points along the trajectory or use the Corner specification.

Syntax

Void SmoothPathMotion(MotionFlags Flags, Axis [] Axes , double [] Point, double Velocity, double ExceptionAngle, double ExceptionLength, double MotionDelay, string Segments);

Arguments

Flags	Bit-mapped argument that can include one or more of the following flags: **ACSC_AMF_WAIT** - Plan the motion but do not start it until the function **acsc_GoM** is executed. **ACSC_AMF_VELOCITY** - The motion will use velocity specified for each segment instead of the default velocity. **ACSC_AMF_DELAY_MOTION** - Defines actual motor movement delay in microseconds. The delay resolution is 50 microseconds. The switch requires an additional parameter that specifies the motion delay. **ACSC_AMF_NURBS_CONSIDER_ACCELERATION** - Acceleration consideration. Allow the motion generator to deviate from specified axes acceleration parameter during velocity profile generation. **ACSC_AMF_NURBS_EXCEPTION_ANGLE** - The suffix requires additional parameter that specifies maximum angle in a control point. The value defines exceptions from spline interpolation. If for an internal control point, directions to the previous and the next control points require direction change more than the specified angle (by modulo), the control point is processed as a corner. Actually, such point divides the spline into two independent splines. **ACSC_AMF_NURBS_EXCEPTION_LENGTH** - Specify exception length. The suffix requires additional parameter that specifies maximum segment length. The value defines exceptions from spline interpolation. If a distance between two control points appears longer than the specified length, the trajectory between the points is considered straight. Actually, two independent splines are built before and after the segment.
Point	Array of the final point coordinates. The number and order of values must correspond to the Axes array. The Point must specify a value for each element of Axes except the last –1 element.
Axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to axis 0, ACSC_AXIS_1 to axis 1, etc. After the last axis, one additional element must be located that contains –1 which marks the end of the array. For the axis constants see Axis Definitions.
Velocity	If the **ACSC_AMF_VELOCITY** flag was specified, this argument specifies a motion velocity for current segment. Set this argument to **ACSC_NONE** if not used.

ExceptionAngle	If the **ACSC_AMF_NURBS_EXCEPTION_ANGLE** flag was specified, this argument defines ExceptionAngle. Set this argument to **ACSC_NONE** if not used.
ExceptionLength	If the **ACSC_AMF_ NURBS_EXCEPTION_LENGTH** flag was specified, this argument defines ExceptionLength. Set this argument to **ACSC_NONE** if not used.
MotionDelay	If the **ACSC_AMF_DELAY_MOTION** flag has been specified, this argument defines the bit motion delay. Defines actual motor movement delay in microseconds. The delay resolution is 50 microseconds. The maximum delay is 100 controller cycles or 100ms for **CTIME**=1ms or 20ms for CTIME=0.2ms. Set this argument to **ACSC_NONE** if not used
Segments	String that contains the name of a one-dimensional user-defined array used to store added segments. Set this argument to NULL if not used. By default, if this argument is not specified, the controller allocates internal buffer for storing 50 segments only. The argument allows the user application to reallocate the buffer for storing a larger number of segments. The larger number of segments may be required if the application needs to add many very small segments in advanced. For most applications, the internal buffer size is enough and should not be enlarged. The buffer is for the controller internal use only and should not be used by the user application. The buffer size calculation rule: each segment requires about 600 bytes, so if it is necessary to allocate the buffer for 200 segments, it should be at least 600 * 200 = 120,000 bytes. The following declaration defines a 120,000 bytes buffer: real buf(15000) See **XARRSIZE** explanation in the ACSPL+ Command and Variable Reference Guide for details on how to declare a buffer with more than 100,000 elements.

Return Value

None

Example

```
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1, Axis.ACSC_NONE };
double[] segment = { 0.0,0.0 };

// Enable motors before using the SPATH motion.
```

```
acsApi.Enable(Axis.ACSC_AXIS_0);
acsApi.WaitMotorEnabled(Axis.ACSC_AXIS_0,1,5000);

acsApi.Enable(Axis.ACSC_AXIS_1);
acsApi.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, 5000);

acsApi.SmoothPathMotion(
                MotionFlags.ACSC_AMF_VELOCITY, // Specified velocity flag
                axes,        // Array of axis
                segment,      // Segment Point start position
                1000,        // Velocity
                -1,          // Exception Angle
                -1,          // Exception Length
                -1,          // Motion Delay
                null         // Segments
                );
```

3.24.4 SmoothPathSegment

Description

Smooth path is a motion type that accepts line segments and generates a spline motion between the specified points.

The motion deviates from exact coordinates after interpolation.

To control the deviation, the user can generate more points along the trajectory or use the Corner specification.

Syntax

void SmoothPathMotion(MotionFlags Flags, Axis [] Axes , double [] Point, double Velocity, double ExceptionAngle, double ExceptionLength, double MotionDelay, string Segments);

Arguments

Flags	Bit-mapped argument that can include one or more of the following flags: **ACSC_AMF_WAIT** - plan the motion but do not start it until the function **acsc_GoM** is executed. **ACSC_AMF_VELOCITY** - the motion will use velocity specified for each segment instead of the default velocity. **ACSC_AMF_ENDVELOCITY** - This flag requires an additional parameter that specifies end velocity. The controller decelerates to the specified velocity at the end of the segment. The specified value should be less than the required velocity; otherwise the parameter is ignored. This flag affects only one segment. This flag also disables corner detection and processing at the end of segment. If this flag is not specified, deceleration is not required. However, in special cases the deceleration might occur due to corner processing or other velocity control conditions. **ACSC_AMF_MAXIMUM** - use maximum velocity under axis limits. With this suffix, no required velocity should be specified. The required velocity is calculated for each segment individually on the base of segment geometry and axis velocities (VEL values) of the involved axes. **ACSC_AMF_JUNCTIONVELOCITY** - Decelerate to corner. This flag requires an additional parameter that specifies corner velocity. The controller detects corner on the path and decelerates to the specified velocity before the corner. The specified value should be less than the required velocity; otherwise the parameter is ignored. If the **ACSC_AMF_JUNCTIONVELOCITY** flag is not specified while the **ACSC_AMF_ANGLE** flag is specified, zero value of corner velocity is assumed. If neither the **ACSC_AMF_JUNCTIONVELOCITY** nor the **ACSC_ AMF_ANGLE** flags are specified, the controller provides automatic calculation as described in Automatic corner processing. **ACSC_AMF_ANGLE** - Do not treat junction as a corner, if junction angle is less than or equal to the specified value in radians. This flag requires additional parameter that specifies a negligible angle in radians. If **ACSC_AMF_ANGLE** flag is not specified while **ACSC_AMF_JUNCTIONVELOCITY** flag is specified, the controller accepts default value of 0.01 radians, about 0.57 degrees. If neither the **ACSC_AMF_JUNCTIONVELOCITY** nor the **ACSC_ AMF_ANGLE** flags are specified, the controller provides automatic calculation as

	described in Automatic corner processing. **ACSC_AMF_AXISLIMIT** - Enable velocity limitations under axis limits. With this flag set, setting the **ACSC_AMF_VELOCITY** flag will result in the requested velocity being constrained by the velocity limits of all involved axes. **ACSC_AMF_CURVEVELOCITY** - Decelerate to curvature discontinuity point. This flag requires an additional parameter that specifies velocity at curvature discontinuity points. Curvature discontinuity occurs in linear-to-arc or arc-to-arc smooth junctions. If the flag is not set, the controller does not decelerate to smooth junction disregarding curvature discontinuity in the junction. If the flag is set, the controller detects curvature discontinuity points on the path and provides deceleration to the specified velocity. The specified value should be less than the required velocity; otherwise the parameter is ignored. The flag can be set together with flags **ACSC_AMF_ JUNCTIONVELOCITY** and/or **ACS_AMF_ANGLE**. If neither of **ACSC_AMF_JUNCTIONVELOCITY**, **ACS_AMF_ ANGLE** or **ACSC_ AMF_CURVEVELOCITY** is set, the controller provides automatic calculation of the corner processing. **ACSC_AMF_CURVEAUTO** - If this flag is specified the controller provides automatic calculations as described in Enhanced automatic corner and curvature discontinuity points processing. **ACSC_AMF_CORNERDEVIATION** - Use a corner rounding option with the specified permitted deviation. This flag requires an additional parameter that specifies maximal allowed deviation of motion trajectory
Axes	Array of axis constants. Each element specifies one involved axis: ACSC_ AXIS_0 corresponds to axis 0, ACSC_AXIS_1 to axis 1, etc. After the last axis, one additional element must be located that contains – 1 which marks the end of the array. For the axis constants see Axis Definitions.
Point	Array of the final point coordinates. The number and order of values must correspond to the Axes array. The Point must specify a value for each element of Axes except the last –1 element.
Velocity	If the **ACSC_AMF_VELOCITY** flag was specified, this argument specifies a motion velocity for current segment. Set this argument to **ACSC_NONE** if not used.

Required_vel	If the **ACSC_AMF_REQUIRED_VELOCITY** flag was specified, this argument specifies a Required velocity in the current point. Set this argument to **ACSC_NONE** if not used.

Return Value

None

Example

```
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1, Axis.ACSC_NONE };
double[] segment = { 0.0,0.0 };
double[][] segmentArray = new double[][] {
                    new double[] { 50, 0.0 },
                    new double[] { 100, 0.0 },
                    new double[] { 100, 100.0 },
                    new double[] { 200, 200 },
                    new double[] { 300, 200 },
                    new double[] { 400, 300 },
                    new double[] { 400, 400 }
                 };

// Enable motors before using the SPATH motion.
acsApi.Enable(Axis.ACSC_AXIS_0);
acsApi.WaitMotorEnabled(Axis.ACSC_AXIS_0,1,5000);

acsApi.Enable(Axis.ACSC_AXIS_1);
acsApi.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, 5000);

acsApi.SmoothPathMotion(
                MotionFlags.ACSC_AMF_VELOCITY, // Specified velocity flag
                axes,       // Array of axis
                segment,     // Segment Point start position
                1000,       // Velocity
                -1,         // Exception Angle
                -1,         // Exception Length
                -1,         // Motion Delay
                null        // Segments
                );

for (int i = 0; i < 7; i++)
{
      segment[0] = segmentArray[i][0];
      segment[1] = segmentArray[i][1];
      acsApi.SmoothPathSegment(
                    MotionFlags.ACSC_NONE, // Motion Flag
                    axes,       // Array of axis
                    segment,     // Segment Point
                    -1,         // Velocity
                    -1          // Required Velocity
```

```
                        );
}
acsApi.EndSequenceM(axes);
```

3.25 Points and Segments Manipulation Methods

The Points and Segments Manipulation methods are:

Table 3-28. Points and Segments Manipulation Methods

Method	Description
AddPoint	Adds a point to a single-axis multi-point or spline motion.
AddPointM	Adds a point to a multi-axis multi-point or spline motion.
ExtAddPoint	Adds a point to a single-axis multi-point or spline motion and specifies a specific velocity or motion time.
ExtAddPointM	Adds a point to a multi-axis multi-point or spline motion and specifies a specific velocity or motion time.
EndSequence	Informs the controller that no more points will be specified for the current single-axis motion.
EndSequenceM	Informs the controller that no more points or segments will be specified for the current multi-axis motion.

3.25.1 AddPoint

Description

The method adds a point to a single axis multi-point or spline motion.

Syntax

object.AddPoint(Axis axis, double point)

Async Syntax

ACSC_WAITBLOCK object.AddPointAsync(Axis axis, double point)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
point	Coordinate of the added point.

Return Value

None

Remarks

The method adds a point to a single-axis multi-point or spline motion. To add a point to a multi- axis motion, use *AddPVPointM*. To add a point with a specified non-default velocity ortime interval use *ExtAddPoint* or *ExtAddPointM*.

The method waits for the controller response.

The controller response indicates that the command was accepted and the point is added to the motion buffer. The point can be rejected if the motion buffer is full. In this case, you can call this method periodically until the method returns a non-zero value.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
double[] points = { 0, 0 };
Ch.Enable(Axis.ACSC_AXIS_0); // Enable axis 0
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Create multi-point motion with default velocity and
// dwell 1 ms
Ch.MultiPoint(MotionFlags.ACSC_NONE, Axis.ACSC_AXIS_0, 1);
for (int index = 0; index < 5; index++)
{
    Ch.AddPoint(Axis.ACSC_AXIS_0, 100 * index);
}
// Finish the motion
// End of the multi-point motion
Ch.EndSequence(Axis.ACSC_AXIS_0);
```

3.25.2 AddPointM

Description

The method adds a point to a multi-axis multi-point or spline motion.

Syntax

object.AddPointM(Axis[] axes, double[] point)

Async Syntax

ACSC_WAITBLOCK object.AddPointMAsync(Axis[] axes, double[] point)

Arguments

axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.
point	Array of the coordinates of added point. The number and order of values must correspond to the **axes** array. **point** must specify a value for each element of **axes** except the last –1 element.

Return Value

None

Remarks

The method adds a point to a multi-axis multi-point or spline motion. To add a point to a single- axis motion, use *AddPoint*. To add a point with a specified non-default velocity or time interval use *ExtAddPoint* or *ExtAddPointM*.

The method waits for the controller response.

The controller response indicates that the command was accepted and the point is added to the motion buffer. The point can be rejected if the motion buffer is full. In this case, you can call this method periodically until the method returns non-zero value.

All axes specified in the axes array must be specified before the call of the *MultiPointM* or *SplineM* method. The number and order of the axes in the axes array must correspond exactly to the number and order of the axes of *MultiPointM* or *SplineM* methods.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1, Axis.ACSC_NONE };
double[] points = { 0, 0 };
Ch.EnableM(axes); // Enable axes 0 and 1
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Wait axis 1 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
// Create multi-point motion with default velocity without dwell
Ch.MultiPointM(MotionFlags.ACSC_NONE, axes, 0);
// Add some points
for (int index = 0; index < 5; index++)
{
     // Position and velocity for each point
     points[0] = 100 * index;
     points[1] = 200 * index;
Ch.AddPointM(axes, points);
}
```

```
// Finish the motion
// End of the multi-point motion
Ch.EndSequenceM(axes);
```

3.25.3 ExtAddPoint

Description

The method adds a point to a single-axis multi-point or spline motion and specifies a specific velocity or motion time.

Syntax

object.ExtAddPoint(Axis axis, double point, double rate)

Async Syntax

ACSC_WAITBLOCK object.ExtAddPointAsync(Axis axis, double point, double rate)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
point	Coordinate of the added point.
rate	If the motion was activated by the MultiPoint method with the **ACSC_AMF_VELOCITY** flag, this parameter defines the motion velocity. If the motion was activated by the Spline method with the **ACSC_AMF_VARTIME** flag, this parameter defines the time interval between the previous point and the present one.

Return Value

None

Remarks

The method adds a point to a single-axis multi-point motion with specific velocity or to single- axis spline motion with a non-uniform time.

To add a point to a multi-axis motion, use *ExtAddPointM*. To add a point to a motion with default velocity or uniform time interval, the *AddPoint* and *AddPointM* methods aremore convenient.

The method waits for the controller response.

The controller response indicates that the command was accepted and the point is added to the motion buffer. The point can be rejected if the motion buffer is full. In this case, you can call this method periodically until the method returns a non-zero value.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Ch.Enable(Axis.ACSC_AXIS_0); // Enable axis 0
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Create multi-point motion, use the velocity specified
// with each point with dwell 1 ms
Ch.MultiPoint(MotionFlags.ACSC_AMF_VELOCITY, Axis.ACSC_AXIS_0,1);
for (int index = 0; index < 5; index++)
{
Ch.ExtAddPoint(Axis.ACSC_AXIS_0, 100 * index, 5000);
}
// Finish the motion
// End of the multi-point motion
Ch.EndSequence(Axis.ACSC_AXIS_0);
```

3.25.4 ExtAddPointM

Description

The method adds a point to a multi-axis multi-point or spline motion and specifies aspecific velocity or motion time.

Syntax

object.ExtAddPointM(Axis[] axes, double[] point, double rate)

Async Syntax

ACSC_WAITBLOCK object.ExtAddPointMAsync(Axis[] axes, double[] point, double rate)

Arguments

axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.
point	Array of the coordinates of added point. The number and order of values must correspond to the **axes** array. The **point** must specify a value for each element of **axes** except the last –1 element.
rate	If the motion was activated by the MultiPoint method with the **ACSC_AMF_VELOCITY** flag, this parameter defines as motion velocity. If the motion was activated by the Spline method with the **ACSC_AMF_VARTIME** flag, this parameter defines as time interval between the previous point and the present one.

Return Value

None

Remarks

The method adds a point to a multi-axis multi-point or spline motion. To add a point to a single- axis motion, use *ExtAddPoint*. To add a point with to a motion with default velocity or uniform time interval the *ExtAddPoint* and *ExtAddPointM* methods are more convenient.

The method waits for the controller response.

The controller response indicates that the command was accepted and the point is added to the motion buffer. The point can be rejected if the motion buffer is full. In this case, you can call this method periodically until the method returns a non-zero value.

All axes specified in the **axes** array must be specified before calling the *MultiPointM* or *SplineM* methods. The number and order of the axes in the **axes** array must correspond exactly to the number and order of the axes of **MultiPointM** or **SplineM** methods.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
    Axis.ACSC_NONE };
double[] points = { 0, 0 };
Ch.EnableM(axes); // Enable axes 0 and 1
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Wait axis 1 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
// Create multi-point motion with default velocity without
// dwell
Ch.MultiPointM(MotionFlags.ACSC_AMF_VELOCITY, axes, 0);
// Add some points
for (int index = 0; index < 5; index++)
{
    // Position and velocity for each point
    points[0] = 100 * index;
    points[1] = 100 * index;
    Ch.ExtAddPointM(axes, points, 5000);
}
// Finish the motion
// End of the multi-point motion
Ch.EndSequenceM(axes);
```

3.25.5 EndSequence

Description

The method informs the controller that no more points will be specified for the current single- axis motion.

Syntax

object.EndSequence(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.EndSequenceAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

None

Remarks

The motion finishes when the EndSequence method is executed. If the call of EndSequence is omitted, the motion will stop at the last point of the sequence and wait for the next point. No transition to the next motion in the motion queue will occur until the EndSequence method executes.The method waits for the controller response.This method applies to the single-axis multi-point or spline (arbitrary path) motions. If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Ch.Enable(Axis.ACSC_AXIS_0); // Enable axis 0
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Create multi-point motion with default velocity and dwell
// 1 ms
Ch.MultiPoint(MotionFlags.ACSC_NONE, Axis.ACSC_AXIS_0, 1);
for (int index = 0; index < 5; index++)
{
    Ch.AddPoint(Axis.ACSC_AXIS_0, 100 * index);
}
// Finish the motion
// End of the multi-point motion
Ch.EndSequence(Axis.ACSC_AXIS_0);
```

3.25.6 EndSequenceM

Description

The method informs the controller that no more points or segments will be specified for the current multi-axis motion.

Syntax

object.EndSequence(Axis[] axes)

Async Syntax

ACSC_WAITBLOCK object.EndSequenceAsync(Axis[] axes)

Arguments

axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.

Return Value

None

Remarks

The motion finishes when the **EndSequenceM** method is executed. If the call of **EndSequenceM** is omitted, the motion will stop at the last point or segment of the sequence and wait for the next point. No transition to the next motion in the motion queue will occur until the **EndSequenceM** method executes.

The method waits for the controller response.

This method applies to the multi-axis multi-point, spline (arbitrary path) and segmented motions. If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1,
    Axis.ACSC_NONE };
double[] points = { 0, 0 };
Ch.EnableM(axes); // Enable axes 0 and 1
// Wait axis 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0, 1, timeout);
// Wait axis 1 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_1, 1, timeout);
// Create multi-point motion with default velocity without
// dwell
Ch.MultiPointM(MotionFlags.ACSC_NONE, axes, 0);
// Add some points
for (int index = 0; index < 5; index++)
{
    // Position and velocity for each point
    points[0] = 100 * index;
    points[1] = 100 * index;
    Ch.AddPointM(axes, points);
}
// Finish the motion
// End of the multi-point motion
Ch.EndSequenceM(axes);
```

3.26 Dynamic Error Compensation

3.26.1 DynamicErrorCompensationOn

Description

This method activates error correction for the mechanical error compensation for the specified zone.

Syntax

public void DynamicErrorCompensationOn(Axis axis, Zone zone)

Async Syntax

public ACSC_WAITBLOCK DynamicErrorCompensationOnAsync(Axis axis, Zone zone)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to axis 0, ACSC_AXIS_1 – to axis 1, etc.
zone	The zone index, valid numbers are: 0, 1, 2, ... up to the maximum number of zones (up to 10) minus 1. If '-1' is specified, all zones of specified axis will be affected.

Return Value

None

3.26.2 DynamicErrorCompensationOff

Description

The method deactivates error mapping correction for the mechanical error compensation for the specified zone.

Syntax

public void DynamicErrorCompensationOff(Axis axis, Zone zone)

Async Syntax

public ACSC_WAITBLOCK DynamicErrorCompensationOffAsync(Axis axis, Zone zone)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to axis 0, ACSC_AXIS_1 – to axis 1, etc.
zone	The zone index, valid numbers are: 0, 1, 2, ... up to the maximum number of zones (up to 10) minus 1. If '-1' is specified, all zones of specified axis will be affected.

Return Value

None

3.26.3 DynamicErrorCompensationRemove

Description

This method deactivates error correction for the mechanical error compensation for the specified zone.

Syntax

public void DynamicErrorCompensationRemove(Axis axis, Zone zone)

Async Syntax

public ACSC_WAITBLOCK DynamicErrorCompensationRemoveAsync(Axis axis, Zone zone)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to axis 0, ACSC_AXIS_1 – to axis 1, etc.
zone	The zone index, valid numbers are: 0, 1, 2, ... up to the maximum number of zones (up to 10) minus 1. If '-1' is specified, all zones of specified axis will be affected.

Return Value

None

3.26.4 DynamicErrorCompensation1D

Description

The **DynamicErrorCompensation1D** function configures and activates 1D error correction for the mechanical error compensation for the specified zone, so that the compensated reference position is calculated by subtracting the linearly (by default) interpolated error from the desired position so that the actual value will be closer to the desired value. The calculation assumes fixed intervals between points inside the zone.

Syntax

public void DynamicErrorCompensation1D(DynamicErrorCompensationFlags flags, Axis axis, Zone zone, double base0, double step0, string correctionMapVariable, int referencedAxisOrAnalogInput)

Async Syntax

public ACSC_WAITBLOCK DynamicErrorCompensation1DAsync(DynamicErrorCompensationFlags flags, Axis axis, Zone zone, double base0, double step0, string correctionMapVariable, int referencedAxisOrAnalogInput)

Arguments

flags	ACSC_DECOMP_PREVENT_COMP_INDEX_MARK_PEG Prevent applying dynamic error compensation on INDEX, MARK, and PEG values ACSC_DECOMP_REFERENCED_AXIS Specifies that the mechanical error compensation is calculated based on the feedback from the axis specified by the optional parameter. ACSC_DECOMP_ANALOG_INPUT Specifies that the mechanical error compensation is calculated based on the feedback from the analog input indicated by the optional parameter. 0, //No flags are set
axis	Axis constant: ACSC_AXIS_0 corresponds to axis 0, ACSC_AXIS_1 – to axis 1, etc. For the axis constants see Axis Definitions.
zone	The zone index, valid numbers are: 0, 1, 2, ... up to the maximum number of zones (10) minus 1. If '-1' is specified, all zones of specified axis will be affected.
base0	A real number representing the axis command that corresponds to the first point in correction table for mechanical error compensation.
step0	A real number representing the fixed interval distance between the two adjacent axis commands.
correctionMapVariable	The name of a real one-dimensional array that specifies correction table for mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
referencedAxisOrAnalogInput	The index of the axis, or the index of the analog input that the mechanical error compensation is calculated based on its feedback.

Return Value

None

3.26.5 DynamicErrorCompensationN1D

Description

The **DynamicErrorCompensationN1D** function configures and activates 1D error correction for the mechanical error compensation for the specified zone, so that the compensated reference position is calculated by subtracting the linearly (by default) interpolated error from the desired position so that the actual value will be closer to the desired value. The calculation is based on an arbitrary network of points inside the zone.

Syntax

public void DynamicErrorCompensationN1D(DynamicErrorCompensationFlags flags, Axis axis, Zone zone, string axisCommands, string correctionMapVariable, int referencedAxisOrAnalogInput)

Async Syntax

public ACSC_WAITBLOCK DynamicErrorCompensationN1DAsync(DynamicErrorCompensationFlags flags, Axis axis, Zone zone, string axisCommands, string correctionMapVariable, int referencedAxisOrAnalogInput)

Arguments

flags	ACSC_DECOMP_PREVENT_COMP_INDEX_MARK_PEG Prevent applying dynamic error compensation on INDEX, MARK, and PEG values ACSC_DECOMP_REFERENCED_AXIS Specifies that the mechanical error compensation is calculated based on the feedback from the axis specified by the optional parameter. ACSC_DECOMP_ANALOG_INPUT Specifies that the mechanical error compensation is calculated based on the feedback from the analog input indicated by the optional parameter. 0, //No flags are set
axis	Axis constant: ACSC_AXIS_0 corresponds to axis 0, ACSC_AXIS_1 – to axis 1, etc. For the axis constants see Axis Definitions.
zone	The zone index, valid numbers are: 0, 1, 2, ... up to the maximum number of zones (10) minus 1. If '-1' is specified, all zones of specified axis will be affected.
axisCommands	The name of a real one-dimensional array that specifies axis command values used for correction table of mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMapVariable	The name of a real one-dimensional array that specifies correction table for mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).

referencedAxisOrAnalogInput	The index of the axis, or the index of the analog input that the mechanical error compensation is calculated based on its feedback.

Return Value

None

3.26.6 DynamicErrorCompensationA1D

Description

The **DynamicErrorCompensationA1D** function configures and activates 1D error correction for the mechanical error compensation for the specified zone, so that the compensated reference position is calculated by multiplying the desired position by a scaling factor ACSLP+so that the actual value will be closer to the desired value.

Syntax

public void DynamicErrorCompensationA1D(DynamicErrorCompensationFlags flags, Axis axis, Zone zone, double scalingFactor, double offset)

Async Syntax

public ACSC_WAITBLOCK DynamicErrorCompensationA1DAsync(DynamicErrorCompensationFlags flags, Axis axis, Zone zone, double scalingFactor, double offset)

Arguments

flags	ACSC_DECOMP_PREVENT_COMP_INDEX_MARK_PEG Prevent applying dynamic error compensation on INDEX, MARK, and PEG values ACSC_DECOMP_REFERENCED_AXIS Specifies that the mechanical error compensation is calculated based on the feedback from the axis specified by the optional parameter. ACSC_DECOMP_ANALOG_INPUT Specifies that the mechanical error compensation is calculated based on the feedback from the analog input indicated by the optional parameter. 0, //No flags are set
axis	Axis constant: ACSC_AXIS_0 corresponds to axis 0, ACSC_AXIS_1 – to axis 1, etc. For the axis constants see Axis Definitions.
zone	The zone index, valid numbers are: 0, 1, 2, ... up to the maximum number of zones (10) minus 1. If '-1' is specified, all zones of specified axis will be affected.
scalingFactor	The scaling factor for the linear alignment that will be used for mechanical error compensation. The allowed range for the scaling factor is 0>2.

offset	The offset for the linear alignment that will be used for mechanical error compensation. The offset is actually the mechanical error compensation for the 0-point location.

Return Value

None

3.26.7 DynamicErrorCompensation2D

Description

The **DynamicErrorCompensation2D** function configures and activates 2D error correction for the mechanical error compensation of the ‘axis0’ command for the specified zone, so that the compensated reference position is calculated by subtracting the linearly (by default) interpolated error from the desired position so that the actual value will be closer to the desired value.

Syntax

public void DynamicErrorCompensation2D(DynamicErrorCompensationFlags flags, int axis0, int axis1, Zone zone, double base0, double step0, double base1, double step1, string correctionMapVariable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1)

Async Syntax

public ACSC_WAITBLOCK DynamicErrorCompensation2DAsync(DynamicErrorCompensationFlags flags, int axis0, int axis1, Zone zone, double base0, double step0, double base1, double step1, string correctionMapVariable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1)

Arguments

flags	ACSC_DECOMP_PREVENT_COMP_INDEX_MARK_PEG Prevent applying dynamic error compensation on INDEX, MARK, and PEG values ACSC_DECOMP_REFERENCED_AXIS Specifies that the mechanical error compensation is calculated based on the feedback from the axis specified by the optional parameter. ACSC_DECOMP_ANALOG_INPUT Specifies that the mechanical error compensation is calculated based on the feedback from the analog input indicated by the optional parameter. 0, //No flags are set
axis	Axis constant: ACSC_AXIS_0 corresponds to axis 0, ACSC_AXIS_1 – to axis 1, etc. For the axis constants see Axis Definitions.

zone	The zone index, valid numbers are: 0, 1, 2, ... up to the maximum number of zones (10) minus 1. If '-1' is specified, all zones of specified axis will be affected.
base0	A real number representing the 'axis0' command that corresponds to the first point in correction table for mechanical error compensation.
step0	A real number representing the fixed interval distance between the two adjacent 'axis0' commands.
base1	A real number representing the 'axis1' command that corresponds to the first point in correction table for mechanical error compensation.
step1	A real number representing the fixed interval distance between the two adjacent 'axis1' commands.
correctionMapVariable	The name of a real one-dimensional array that specifies correction table for mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
referencedAxisOrAnalogInput0	The index of the first axis, or the index of the first analog input whose feedback will be used to calculate the mechanical error compensation.
referencedAxisOrAnalogInput1	The index of the second axis, or the index of the second analog input whose feedback will be used to calculate the mechanical error compensation.

Return Value

None

3.26.8 DynamicErrorCompensationN2D

Description

The **DynamicErrorCompensationN2D** function configures and activates 2D error correction for the mechanical error compensation of the 'axis0' parameter for the specified zone, so that the compensated reference position is calculated by subtracting the linearly (by default) interpolated error from the desired position so that the actual value will be closer to the desired value. The calculation is based on an arbitrary network of points inside the zone.

Syntax

public void DynamicErrorCompensationN2D(DynamicErrorCompensationFlags flags, int axis0, int axis1, Zone zone, string axis0Commands, string axis1Commands, string correctionMapVariable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1)

Async Syntax

public ACSC_WAITBLOCK DynamicErrorCompensationN2DAsync(DynamicErrorCompensationFlags flags, int axis0, int axis1, Zone zone, string axis0Commands, string axis1Commands, string correctionMapVariable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1)

Arguments

flags	ACSC_DECOMP_PREVENT_COMP_INDEX_MARK_PEG Prevent applying dynamic error compensation on INDEX, MARK, and PEG values ACSC_DECOMP_REFERENCED_AXIS Specifies that the mechanical error compensation is calculated based on the feedback from the axis specified by the optional parameter. ACSC_DECOMP_ANALOG_INPUT Specifies that the mechanical error compensation is calculated based on the feedback from the analog input indicated by the optional parameter. 0, //No flags are set
axis 0	The index of the first axis that the mechanical error compensation will be applied to. Valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
axis 1	The index of the second axis that the mechanical error compensation will be applied to. Valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
zone	The zone index, valid numbers are: 0, 1, 2, ... up to the maximum number of zones (10) minus 1. If '-1' is specified, all zones of specified axis will be affected.
axis0Commands	The name of a real one-dimensional array that specifies 'axis0' command values used for correction table of mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
axis1Commands	The name of a real one-dimensional array that specifies 'axis1' command values used for correction table of mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMapVariable	The name of a real one-dimensional array that specifies correction table for mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).

referencedAxisOrAnalogInput0	The index of the first axis, or the index of the first analog input whose feedback will be used to calculate the mechanical error compensation.
referencedAxisOrAnalogInput1	The index of the second axis, or the index of the second analog input whose feedback will be used to calculate the mechanical error compensation.

Return Value

None

3.26.9 DynamicErrorCompensationA2D

Description

The acsc_DynamicErrorCompensationA2D function configures and activates 2D error correction for the mechanical error compensation of the specified axis for the specified zone, so that the compensated reference position is calculated by taking into account the angle for the orthogonality correction so that the actual value will be closer to the desired value.

Syntax

public void DynamicErrorCompensationA2D(DynamicErrorCompensationFlags flags, int axis0, int axis1, Zone zone, double angle)

Async Syntax

public ACSC_WAITBLOCK DynamicErrorCompensationA2DAsync(DynamicErrorCompensationFlags flags, int axis0, int axis1, Zone zone, double angle)

Arguments

flags	ACSC_DECOMP_PREVENT_COMP_INDEX_MARK_PEG Prevent applying dynamic error compensation on INDEX, MARK, and PEG values ACSC_DECOMP_REFERENCED_AXIS Specifies that the mechanical error compensation is calculated based on the feedback from the axis specified by the optional parameter. ACSC_DECOMP_ANALOG_INPUT Specifies that the mechanical error compensation is calculated based on the feedback from the analog input indicated by the optional parameter. 0, //No flags are set
axis 0	The index of the first axis that the mechanical error compensation will be applied to. Valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
axis 1	The index of the second axis that the mechanical error compensation will be applied to. Valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.

zone	The zone index, valid numbers are: 0, 1, 2, ... up to the maximum number of zones (10) minus 1. If '-1' is specified, all zones of specified axis will be affected.
angle	The angle for the orthogonality correction that will be used for mechanical error compensation. The allowed range for the angle is [-45°, 45°].

Return Value

None

3.26.10 DynamicErrorCompensation3D2

Description

The **DynamicErrorCompensation3D2** function configures and activates 3D error correction for the mechanical error compensation of the 'axis0', 'axis1', and 'axis2' parameters for the specified zone, so that the compensated reference position is calculated by adding the linearly (by default) interpolated error from the desired position so that the actual value will be closer to the desired value.

Syntax

public void DynamicErrorCompensation3D2(DynamicErrorCompensationFlags flags, int axis0, int axis1, int axis2, Zone zone, double base0, double step0, double base1, double step1, double base2, double step2, string correctionMap0Variable, string correctionMap1Variable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1, int referencedAxisOrAnalogInput2)

Async Syntax

public ACSC_WAITBLOCK DynamicErrorCompensation3D2Async(DynamicErrorCompensationFlags flags, int axis0, int axis1, int axis2, Zone zone, double base0, double step0, double base1, double step1, double base2, double step2, string correctionMap0Variable, string correctionMap1Variable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1, int referencedAxisOrAnalogInput2)

Arguments

Flags	ACSC_DECOMP_00 The mechanical error compensation will be applied to ‘axis0’ (default) ACSC_DECOMP_01 The mechanical error compensation will be applied to ‘axis1’ ACSC_DECOMP_02 The mechanical error compensation will be applied to ‘axis1’ ACSC_DECOMP_REFERENCED_AXIS Specifies that the mechanical error compensation is calculated based on the feedback from the axis specified by the optional parameter. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_FIRST_ANALOG_INPUT Specifies that the first optional parameter will be regarded as an analog input index. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_SECOND_ANALOG_INPUT Specifies that the second optional parameter will be regarded as an analog input index. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_THIRD_ANALOG_INPUT Specifies that the third optional parameter will be regarded as an analog input index. 0, //No flags are set
axis0	The index of the first axis that the mechanical error compensation will be applied to. Valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
axis1	The index of the second axis participating in 3D mechanical error compensation valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.1.
axis2	The index of the second axis participating in 3D mechanical error compensation valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.1.
Zone	The zone index, valid numbers are: 0, 1, 2, ... up to the maximum number of zones (10) minus 1. If ‘-1’ is specified, all zones of specified axis will be affected.

base0	A real number representing the 'axis0' command that corresponds to the first point in correction table for mechanical error compensation.
step0	A real number representing the fixed interval distance between the two adjacent 'axis0' commands.
base1	A real number representing the 'axis1' command that corresponds to the first point in correction table for mechanical error compensation.
step1	A real number representing the fixed interval distance between the two adjacent 'axis1' commands.
base2	A real number representing the 'axis2' command that corresponds to the first point in correction table for mechanical error compensation.
step2	A real number representing the fixed interval distance between the two adjacent 'axis2' commands.
correctionMap0Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '0' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap1Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '1' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
referencedAxisOrAnalogInput0	The index of the first axis, or the index of the first analog input whose feedback will be used to calculate the mechanical error compensation.
referencedAxisOrAnalogInput1	The index of the second axis, or the index of the second analog input whose feedback will be used to calculate the mechanical error compensation.
referencedAxisOrAnalogInput2	The index of the third axis, or the index of the third analog input whose feedback will be used to calculate the mechanical error compensation.

Return Value

None

3.26.11 DynamicErrorCompensation3D3

Description

The **DynamicErrorCompensation3D3** function configures and activates 3D error correction for the mechanical error compensation of the ‘axis0’, ‘axis1’, and 'axis2' parameters for the specified zone, so that the compensated reference position is calculated by adding the linearly (by default) interpolated error from the desired position so that the actual value will be closer to the desired value.

Syntax

public void DynamicErrorCompensation3D3(DynamicErrorCompensationFlags flags, int axis0, int axis1, int axis2, Zone zone, double base0, double step0, double base1, double step1, double base2, double step2, string correctionMap0Variable, string correctionMap1Variable, string correctionMap2Variable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1, int referencedAxisOrAnalogInput2)

Async Syntax

public ACSC_WAITBLOCK DynamicErrorCompensation3D3Async(DynamicErrorCompensationFlags flags, int axis0, int axis1, int axis2, Zone zone, double base0, double step0, double base1, double step1, double base2, double step2, string correctionMap0Variable, string correctionMap1Variable, string correctionMap2Variable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1, int referencedAxisOrAnalogInput2)

Arguments

Flags	ACSC_DECOMP_00 The mechanical error compensation will be applied to ‘axis0’ (default) ACSC_DECOMP_01 The mechanical error compensation will be applied to ‘axis1’ ACSC_DECOMP_02 The mechanical error compensation will be applied to ‘axis1’ ACSC_DECOMP_REFERENCED_AXIS Specifies that the mechanical error compensation is calculated based on the feedback from the axis specified by the optional parameter. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_FIRST_ANALOG_INPUT Specifies that the first optional parameter will be regarded as an analog input index. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_SECOND_ANALOG_INPUT Specifies that the second optional parameter will be regarded as an analog input index. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_THIRD_ANALOG_INPUT Specifies that the third optional parameter will be regarded as an analog input index. 0, //No flags are set
axis0	The index of the first axis that the mechanical error compensation will be applied to. Valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
axis1	The index of the second axis participating in 3D mechanical error compensation valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.1.
axis2	The index of the third axis participating in 3D mechanical error compensation valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.1.
zone	The zone index, valid numbers are: 0, 1, 2, ... up to the maximum number of zones (10) minus 1. If ‘-1’ is specified, all zones of specified axis will be affected.
base0	A real number representing the ‘axis0’ command that corresponds to the first point in correction table for mechanical error compensation.

step0	A real number representing the fixed interval distance between the two adjacent 'axis0' commands.
base1	A real number representing the 'axis1' command that corresponds to the first point in correction table for mechanical error compensation.
step1	A real number representing the fixed interval distance between the two adjacent 'axis1' commands.
base2	A real number representing the 'axis2' command that corresponds to the first point in correction table for mechanical error compensation.
step2	A real number representing the fixed interval distance between the two adjacent 'axis2' commands.
correctionMap0Variable	The name of a real two-dimensional array that specifies the correction table for mechanical error compensation in relation to the axis 2 step '0' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap1Variable	The name of a real two-dimensional array that specifies the correction table for mechanical error compensation in relation to the axis 2 step '1' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap2Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '2' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
referencedAxisOrAnalogInput0	The index of the first axis, or the index of the first analog input whose feedback will be used to calculate the mechanical error compensation.
referencedAxisOrAnalogInput1	The index of the second axis, or the index of the second analog input whose feedback will be used to calculate the mechanical error compensation.
referencedAxisOrAnalogInput2	The index of the third axis, or the index of the third analog input whose feedback will be used to calculate the mechanical error compensation.

Return Value

None

3.26.12 DynamicErrorCompensation3D5

Description

The **DynamicErrorCompensation3D5** function configures and activates 3D error correction for the mechanical error compensation of the ‘axis0’, ‘axis1’, and 'axis2' parameters for the specified zone, so that the compensated reference position is calculated by adding the linearly (by default) interpolated error from the desired position so that the actual value will be closer to the desired value.

Syntax

public void DynamicErrorCompensation3D5(DynamicErrorCompensationFlags flags, int axis0, int axis1, int axis2, Zone zone, double base0, double step0, double base1, double step1, double base2, double step2, string correctionMap0Variable, string correctionMap1Variable, string correctionMap2Variable, string correctionMap3Variable, string correctionMap4Variable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1, int referencedAxisOrAnalogInput2)

Async Syntax

public ACSC_WAITBLOCK DynamicErrorCompensation3D5Async(DynamicErrorCompensationFlags flags, int axis0, int axis1, int axis2, Zone zone, double base0, double step0, double base1, double step1, double base2, double step2, string correctionMap0Variable, string correctionMap1Variable, string correctionMap2Variable, string correctionMap3Variable, string correctionMap4Variable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1, int referencedAxisOrAnalogInput2)

Arguments

Flags	ACSC_DECOMP_00 The mechanical error compensation will be applied to ‘axis0’ (default) ACSC_DECOMP_01 The mechanical error compensation will be applied to ‘axis1’ ACSC_DECOMP_02 The mechanical error compensation will be applied to ‘axis1’ ACSC_DECOMP_REFERENCED_AXIS Specifies that the mechanical error compensation is calculated based on the feedback from the axis specified by the optional parameter. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_FIRST_ANALOG_INPUT Specifies that the first optional parameter will be regarded as an analog input index. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_SECOND_ANALOG_INPUT Specifies that the second optional parameter will be regarded as an analog input index. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_THIRD_ANALOG_INPUT Specifies that the third optional parameter will be regarded as an analog input index. 0, //No flags are set
axis0	The index of the first axis that the mechanical error compensation will be applied to. Valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
axis1	The index of the second axis participating in 3D mechanical error compensation valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.1.
axis2	The index of the second axis participating in 3D mechanical error compensation valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.1.
zone	The zone index, valid numbers are: 0, 1, 2, ... up to the maximum number of zones (10) minus 1. If ‘-1’ is specified, all zones of specified axis will be affected.
base0	A real number representing the ‘axis0’ command that corresponds to the first point in correction table for mechanical error compensation.

step0	A real number representing the fixed interval distance between the two adjacent ‘axis0’ commands.
base1	A real number representing the ‘axis1’ command that corresponds to the first point in correction table for mechanical error compensation.
step1	A real number representing the fixed interval distance between the two adjacent ‘axis1’ commands.
base2	A real number representing the ‘axis2’ command that corresponds to the first point in correction table for mechanical error compensation.
step2	A real number representing the fixed interval distance between the two adjacent ‘axis2’ commands.
correctionMap0Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step ‘0’ coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap1Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step ‘1’ coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap2Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step ‘2’ coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap3Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step ‘3’ coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap4Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step ‘4’ coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).

referencedAxisOrAnalogInput0	The index of the first axis, or the index of the first analog input whose feedback will be used to calculate the mechanical error compensation.
referencedAxisOrAnalogInput1	The index of the second axis, or the index of the second analog input whose feedback will be used to calculate the mechanical error compensation.
referencedAxisOrAnalogInput2	The index of the third axis, or the index of the third analog input whose feedback will be used to calculate the mechanical error compensation.

Return Value

None

3.26.13 DynamicErrorCompensation3DA

Description

The **DynamicErrorCompensation3DA** function configures and activates 3D error correction for the mechanical error compensation of the ‘axis0’, ‘axis1’, and 'axis2' parameters for the specified zone, so that the compensated reference position is calculated by adding the linearly (by default) interpolated error from the desired position so that the actual value will be closer to the desired value.

Syntax

public void DynamicErrorCompensation3DA(DynamicErrorCompensationFlags flags, int axis0, int axis1, int axis2, Zone zone, double base0, double step0, double base1, double step1, double base2, double step2, string correctionMap0Variable, string correctionMap1Variable, string correctionMap2Variable, string correctionMap3Variable, string correctionMap4Variable, string correctionMap5Variable, string correctionMap6Variable, string correctionMap7Variable, string correctionMap8Variable, string correctionMap9Variable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1, int referencedAxisOrAnalogInput2)

Async Syntax

public ACSC_WAITBLOCK DynamicErrorCompensation3DAAsync(DynamicErrorCompensationFlags flags, int axis0, int axis1, int axis2, Zone zone, double base0, double step0, double base1, double step1, double base2, double step2, string correctionMap0Variable, string correctionMap1Variable, string correctionMap2Variable, string correctionMap3Variable, string correctionMap4Variable, string correctionMap5Variable, string correctionMap6Variable, string correctionMap7Variable, string correctionMap8Variable, string correctionMap9Variable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1, int referencedAxisOrAnalogInput2)

Arguments

Flags	ACSC_DECOMP_00 The mechanical error compensation will be applied to 'axis0' (default) ACSC_DECOMP_01 The mechanical error compensation will be applied to 'axis1' ACSC_DECOMP_02 The mechanical error compensation will be applied to 'axis1' ACSC_DECOMP_REFERENCED_AXIS Specifies that the mechanical error compensation is calculated based on the feedback from the axis specified by the optional parameter. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_FIRST_ANALOG_INPUT Specifies that the first optional parameter will be regarded as an analog input index. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_SECOND_ANALOG_INPUT Specifies that the second optional parameter will be regarded as an analog input index. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_THIRD_ANALOG_INPUT Specifies that the third optional parameter will be regarded as an analog input index. 0, //No flags are set
axis0	The index of the first axis that the mechanical error compensation will be applied to. Valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
axis1	The index of the second axis participating in 3D mechanical error compensation valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.1.
axis2	The index of the second axis participating in 3D mechanical error compensation valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.1.
zone	The zone index, valid numbers are: 0, 1, 2, ... up to the maximum number of zones (10) minus 1. If '-1' is specified, all zones of specified axis will be affected.
base0	A real number representing the 'axis0' command that corresponds to the first point in correction table for mechanical error compensation.

step0	A real number representing the fixed interval distance between the two adjacent 'axis0' commands.
base1	A real number representing the 'axis1' command that corresponds to the first point in correction table for mechanical error compensation.
step1	A real number representing the fixed interval distance between the two adjacent 'axis1' commands.
base2	A real number representing the 'axis2' command that corresponds to the first point in correction table for mechanical error compensation.
step2	A real number representing the fixed interval distance between the two adjacent 'axis2' commands.
correctionMap0Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '0' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap1Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '1' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap2Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '2' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap3Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '3' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap4Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '4' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).

correctionMap5Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '5' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap6Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '6' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap7Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '7' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap8Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '8' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap9Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '9' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
referenced_axis_or_analog_input0	The index of the first axis, or the index of the first analog input whose feedback will be used to calculate the mechanical error compensation.
referencedAxisOrAnalogInput0	The index of the second axis, or the index of the second analog input whose feedback will be used to calculate the mechanical error compensation.
referenced_axis_or_analog_input2	The index of the third axis, or the index of the third analog input whose feedback will be used to calculate the mechanical error compensation.

Return Value

None

3.26.14 DynamicErrorCompensationN3D2

Description

The **DynamicErrorCompensationN3D2** function configures and activates 3D error correction for the mechanical error compensation of the ‘axis0’, ‘axis1’, and 'axis2' parameters for the specified zone, so that the compensated reference position is calculated by adding the linearly (by default) interpolated error from the desired position so that the actual value will be closer to the desired value.

Syntax

public void DynamicErrorCompensationN3D2(DynamicErrorCompensationFlags flags, int axis0, int axis1, int axis2, Zone zone, string axis0Commands, string axis1Commands, string axis2Commands, string correctionMap0Variable, string correctionMap1Variable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1, int referencedAxisOrAnalogInput2)

Async Syntax

public ACSC_WAITBLOCK DynamicErrorCompensationN3D2Async(DynamicErrorCompensationFlags flags, int axis0, int axis1, int axis2, Zone zone, string axis0Commands, string axis1Commands, string axis2Commands, string correctionMap0Variable, string correctionMap1Variable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1, int referencedAxisOrAnalogInput2)

Arguments

Flags	ACSC_DECOMP_00 The mechanical error compensation will be applied to ‘axis0’ (default) ACSC_DECOMP_01 The mechanical error compensation will be applied to ‘axis1’ ACSC_DECOMP_02 The mechanical error compensation will be applied to ‘axis1’ ACSC_DECOMP_REFERENCED_AXIS Specifies that the mechanical error compensation is calculated based on the feedback from the axis specified by the optional parameter. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_FIRST_ANALOG_INPUT Specifies that the first optional parameter will be regarded as an analog input index. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_SECOND_ANALOG_INPUT Specifies that the second optional parameter will be regarded as an analog input index. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_THIRD_ANALOG_INPUT Specifies that the third optional parameter will be regarded as an analog input index. 0, //No flags are set
axis0	The index of the first axis that the mechanical error compensation will be applied to. Valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
axis1	The index of the second axis participating in 3D mechanical error compen valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.stem minus 1.
axis2	The index of the second axis participating in 3D mechanical error compen valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.stem minus 1.
zone	The zone index, valid numbers are: 0, 1, 2, ... up to the maximum number of zones (10) minus 1. If ‘-1’ is specified, all zones of specified axis will be affected.

axis0Commands	The name of a real one-dimensional array that specifies 'axis0' command values used for correction table of mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
axis1Commands	The name of a real one-dimensional array that specifies 'axis1' command values used for correction table of mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
axis2_command	The name of a real one-dimensional array that specifies 'axis2' command values used for correction table of mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
axis2Commands	The name of a real two-dimensional array that specifies a correction table for mechanical error compensation in relation to the first specified coordinate of the Z-axis. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap0Variable	The name of a real two-dimensional array that specifies a correction table for mechanical error compensation in relation to the first specified coordinate of the Z-axis. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap1Variable	The name of a real two-dimensional array that specifies a correction table for mechanical error compensation in relation to the second specified coordinate of the Z-axis. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
referencedAxisOrAnalogInput0	The index of the first axis, or the index of the first analog input whose feedback will be used to calculate the mechanical error compensation.
referencedAxisOrAnalogInput1	The index of the second axis, or the index of the second analog input whose feedback will be used to calculate the mechanical error compensation.
referencedAxisOrAnalogInput2	The index of the third axis, or the index of the third analog input whose feedback will be used to calculate the mechanical error compensation.

Return Value

None

3.26.15 DynamicErrorCompensationN3D3

Description

The **DynamicErrorCompensationN3D3** function configures and activates 3D error correction for the mechanical error compensation of the ‘axis0’, ‘axis1’, and 'axis2' parameters for the specified zone, so that the compensated reference position is calculated by adding the linearly (by default) interpolated error from the desired position so that the actual value will be closer to the desired value.

Syntax

public void DynamicErrorCompensationN3D3(DynamicErrorCompensationFlags flags, int axis0, int axis1, int axis2, Zone zone, string axis0Commands, string axis1Commands, string axis2Commands, string correctionMap0Variable, string correctionMap1Variable, string correctionMap2Variable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1, int referencedAxisOrAnalogInput2)

Async Syntax

public ACSC_WAITBLOCK DynamicErrorCompensationN3D3Async(DynamicErrorCompensationFlags flags, int axis0, int axis1, int axis2, Zone zone, string axis0Commands, string axis1Commands, string axis2Commands, string correctionMap0Variable, string correctionMap1Variable, string correctionMap2Variable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1, int referencedAxisOrAnalogInput2)

Arguments

Flags	ACSC_DECOMP_00 The mechanical error compensation will be applied to 'axis0' (default) ACSC_DECOMP_01 The mechanical error compensation will be applied to 'axis1' ACSC_DECOMP_02 The mechanical error compensation will be applied to 'axis1' ACSC_DECOMP_REFERENCED_AXIS Specifies that the mechanical error compensation is calculated based on the feedback from the axis specified by the optional parameter. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_FIRST_ANALOG_INPUT Specifies that the first optional parameter will be regarded as an analog input index. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_SECOND_ANALOG_INPUT Specifies that the second optional parameter will be regarded as an analog input index. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_THIRD_ANALOG_INPUT Specifies that the third optional parameter will be regarded as an analog input index. 0, //No flags are set
axis0	The index of the first axis that the mechanical error compensation will be applied to. Valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
axis1	The index of the second axis participating in 3D mechanical error compensation valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.1.
axis2	The index of the second axis participating in 3D mechanical error compensation valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.1.
zone	The zone index, valid numbers are: 0, 1, 2, ... up to the maximum number of zones (10) minus 1. If '-1' is specified, all zones of specified axis will be affected.

axis0Commands	The name of a real one-dimensional array that specifies 'axis0' command values used for correction table of mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
axis1Commands	The name of a real one-dimensional array that specifies 'axis1' command values used for correction table of mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
axis2Commands	The name of a real one-dimensional array that specifies 'axis2' command values used for correction table of mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap0Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '0' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap1Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '1' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap2Variable	The name of a real two-dimensional array that specifies a correction table for mechanical error compensation in relation to the third specified coordinate of the Z-axis. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
referencedAxisOrAnalogInput0	The index of the first axis, or the index of the first analog input whose feedback will be used to calculate the mechanical error compensation.
referencedAxisOrAnalogInput1	The index of the second axis, or the index of the second analog input whose feedback will be used to calculate the mechanical error compensation.
referencedAxisOrAnalogInput2	The index of the third axis, or the index of the third analog input whose feedback will be used to calculate the mechanical error compensation.

Return Value

None

3.26.16 DynamicErrorCompensationN3D5

Description

The **DynamicErrorCompensationN3D5** function configures and activates 3D error correction for the mechanical error compensation of the ‘axis0’, ‘axis1’, and 'axis2' parameters for the specified zone, so that the compensated reference position is calculated by adding the linearly (by default) interpolated error from the desired position so that the actual value will be closer to the desired value.

Syntax

public void DynamicErrorCompensationN3D5(DynamicErrorCompensationFlags flags, int axis0, int axis1, int axis2, Zone zone, string axis0Commands, string axis1Commands, string axis2Commands, string correctionMap0Variable, string correctionMap1Variable, string correctionMap2Variable, string correctionMap3Variable, string correctionMap4Variable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1, int referencedAxisOrAnalogInput2)

Async Syntax

public ACSC_WAITBLOCK DynamicErrorCompensationN3D5Async(DynamicErrorCompensationFlags flags, int axis0, int axis1, int axis2, Zone zone, string axis0Commands, string axis1Commands, string axis2Commands, string correctionMap0Variable, string correctionMap1Variable, string correctionMap2Variable, string correctionMap3Variable, string correctionMap4Variable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1, int referencedAxisOrAnalogInput2)

Arguments

flags	ACSC_DECOMP_00 The mechanical error compensation will be applied to 'axis0' (default) ACSC_DECOMP_01 The mechanical error compensation will be applied to 'axis1' ACSC_DECOMP_02 The mechanical error compensation will be applied to 'axis1' ACSC_DECOMP_REFERENCED_AXIS Specifies that the mechanical error compensation is calculated based on the feedback from the axis specified by the optional parameter. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_FIRST_ANALOG_INPUT Specifies that the first optional parameter will be regarded as an analog input index. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_SECOND_ANALOG_INPUT Specifies that the second optional parameter will be regarded as an analog input index. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_THIRD_ANALOG_INPUT Specifies that the third optional parameter will be regarded as an analog input index. 0, //No flags are set
axis0	The index of the first axis that the mechanical error compensation will be applied to. Valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
axis1	The index of the second axis participating in 3D mechanical error compensation valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.1.
axis2	The index of the second axis participating in 3D mechanical error compensation valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.1.
zone	The zone index, valid numbers are: 0, 1, 2, ... up to the maximum number of zones (10) minus 1. If '-1' is specified, all zones of specified axis will be affected.

axis0Commands	The name of a real one-dimensional array that specifies 'axis0' command values used for correction table of mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
axis1Commands	The name of a real one-dimensional array that specifies 'axis1' command values used for correction table of mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
axis2Commands	The name of a real one-dimensional array that specifies 'axis2' command values used for correction table of mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap0Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '0' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap1Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '1' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap2Variable	The name of a real two-dimensional array that specifies a correction table for mechanical error compensation in relation to the third specified coordinate of the Z-axis. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap3Variable	The name of a real two-dimensional array that specifies a correction table for mechanical error compensation in relation to the fourth specified coordinate of the Z-axis. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap4Variable	The name of a real two-dimensional array that specifies a correction table for mechanical error compensation in relation to the fifth specified coordinate of the Z-axis. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).

referencedAxisOrAnalogInput0	The index of the first axis, or the index of the first analog input whose feedback will be used to calculate the mechanical error compensation.
referencedAxisOrAnalogInput1	The index of the second axis, or the index of the second analog input whose feedback will be used to calculate the mechanical error compensation.
referencedAxisOrAnalogInput2	The index of the third axis, or the index of the third analog input whose feedback will be used to calculate the mechanical error compensation.
Wait	Pointer to ACSC_WAITBLOCK structure. If Wait is ACSC_SYNCHRONOUS, the function returns when the controller response is received. If Wait points to a valid ACSC_WAITBLOCK structure, the function returns immediately. The calling thread must then call the acsc_ WaitForAsyncCall function to retrieve the operation result. If Wait is ACSC_IGNORE, the function returns immediately. In this case, the operation result is ignored by the library and cannot be retrieved to the calling thread.

Return Value

None

3.26.17 DynamicErrorCompensationN3DA

Description

The **DynamicErrorCompensationN3DA** function configures and activates 3D error correction for the mechanical error compensation of the ‘axis0’ , ‘axis1’ , and 'axis2' parameters for the specified zone, so that the compensated reference position is calculated by adding the linearly (by default) interpolated error from the desired position so that the actual value will be closer to the desired value.

Syntax

public void DynamicErrorCompensationN3DA(DynamicErrorCompensationFlags flags, int axis0, int axis1, int axis2, Zone zone, string axis0Commands, string axis1Commands, string axis2Commands, string correctionMap0Variable, string correctionMap1Variable, string correctionMap2Variable, string correctionMap3Variable, string correctionMap4Variable, string correctionMap5Variable, string correctionMap6Variable, string correctionMap7Variable, string correctionMap8Variable, string correctionMap9Variable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1, int referencedAxisOrAnalogInput2)

Async Syntax

public ACSC_WAITBLOCK DynamicErrorCompensationN3DAAsync (DynamicErrorCompensationFlags flags, int axis0, int axis1, int axis2, Zone zone, string axis0Commands, string axis1Commands, string axis2Commands, string correctionMap0Variable, string correctionMap1Variable, string correctionMap2Variable, string correctionMap3Variable, string correctionMap4Variable, string correctionMap5Variable, string correctionMap6Variable, string correctionMap7Variable, string correctionMap8Variable, string correctionMap9Variable, int referencedAxisOrAnalogInput0, int referencedAxisOrAnalogInput1, int referencedAxisOrAnalogInput2)

Arguments

flags	ACSC_DECOMP_00 The mechanical error compensation will be applied to ‘axis0’ (default) ACSC_DECOMP_01 The mechanical error compensation will be applied to ‘axis1’ ACSC_DECOMP_02 The mechanical error compensation will be applied to ‘axis1’ ACSC_DECOMP_REFERENCED_AXIS Specifies that the mechanical error compensation is calculated based on the feedback from the axis specified by the optional parameter. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_FIRST_ANALOG_INPUT Specifies that the first optional parameter will be regarded as an analog input index. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_SECOND_ANALOG_INPUT Specifies that the second optional parameter will be regarded as an analog input index. ACSC_DECOMP_REFERENCED_AXIS | ACSC_DECOMP_THIRD_ANALOG_INPUT Specifies that the third optional parameter will be regarded as an analog input index. 0, //No flags are set
axis0	The index of the first axis that the mechanical error compensation will be applied to. Valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
axis1	The index of the second axis participating in 3D mechanical error compensation valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.1.
axis2	The index of the second axis participating in 3D mechanical error compensation valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.1.

zone	The zone index, valid numbers are: 0, 1, 2, ... up to the maximum number of zones (10) minus 1. If '-1' is specified, all zones of specified axis will be affected.
axis0Commands	The name of a real one-dimensional array that specifies 'axis0' command values used for correction table of mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
axis1Commands	The name of a real one-dimensional array that specifies 'axis1' command values used for correction table of mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
axis2Commands	The name of a real one-dimensional array that specifies 'axis2' command values used for correction table of mechanical error compensation. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap0Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '0' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap1Variable	The name of a real two-dimensional array that specifies correction table for mechanical error compensation in relation to axis 2 step '1' coordinate. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap2Variable	The name of a real two-dimensional array that specifies a correction table for mechanical error compensation in relation to the third specified coordinate of the Z-axis. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap3Variable	The name of a real two-dimensional array that specifies a correction table for mechanical error compensation in relation to the fourth specified coordinate of the Z-axis. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).

correctionMap4Variable	The name of a real two-dimensional array that specifies a correction table for mechanical error compensation in relation to the fifth specified coordinate of the Z-axis. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap5Variable	The name of a real two-dimensional array that specifies a correction table for mechanical error compensation in relation to the sixth specified coordinate of the Z-axis. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap6Variable	The name of a real two-dimensional array that specifies a correction table for mechanical error compensation in relation to the seventh specified coordinate of the Z-axis. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap7Variable	The name of a real two-dimensional array that specifies a correction table for mechanical error compensation in relation to the eighth specified coordinate of the Z-axis. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap8Variable	The name of a real two-dimensional array that specifies a correction table for mechanical error compensation in relation to the ninth specified coordinate of the Z-axis. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
correctionMap9Variable	The name of a real two-dimensional array that specifies a correction table for mechanical error compensation in relation to the tenth specified coordinate of the Z-axis. The array type should be GLOBAL REAL STATIC (defined in D-Buffer).
referencedAxisOrAnalogInput0	The index of the first axis, or the index of the first analog input whose feedback will be used to calculate the mechanical error compensation.
referencedAxisOrAnalogInput1	The index of the second axis, or the index of the second analog input whose feedback will be used to calculate the mechanical error compensation.
referencedAxisOrAnalogInput2	The index of the third axis, or the index of the third analog input whose feedback will be used to calculate the mechanical error compensation.

Return Value

None

3.27 Data Collection Methods

The Data Collection methods are:

Table 3-29. Data Collection Methods

Method	Description
DataCollectionExt	Initiates data collection.
StopCollect	Terminates data collection.

3.27.1 DataCollectionExt

Description

The method initiates data collection.

Syntax

object.DataCollectionExt(DataCollectionFlags flags, [Axis axis,] string array, int nSample, double period, string vars)

Async Syntax

ACSC_WAITBLOCK object.DataCollectionExtAsync(DataCollectionFlags flags, [Axis axis,] string array, int nSample, double period, string vars)

Arguments

flags	Bit-mapped parameter that can include one or more of the following flags: **ACSC_DCF_SYNC** – Start data collection synchronously to a motion. **ACSC_DCF_ WAIT** – Create the synchronous data collection, but do not start until the Go method is called. This flag can only be used with the ACSC_DCF_ SYNC flag. **ACSC_DCF_TEMPORAL** – Temporal data collection. The sampling period is calculated automatically according to the collection time. **ACSC_DCF_CYCLIC** – Cyclic data collection uses the collection array as a cyclic buffer and continues infinitely. When the array is full, each new sample overwrites the oldest sample in the array.
axis	Axis constant of the axis to which the data collection must be synchronized: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions. This argument is required only for axis data collection (**ACSC_DCF_SYNC** flag).

array	Name of the array that stores the collected samples. The array must be declared as a global variable by an ACSPL+ program or by the DeclareVariable method.
nSample	Number of samples to be collected.
period	Sampling period in milliseconds. If the **ACSC_DCF_TEMPORAL** flag is specified, this parameter defines a minimum period.
vars	The string contains chained names of the variables, separated by '\r'(13) character. The values of these variables will be collected in the **array**. If variable name specifies an array, the name must be supplemented with indexes in order to specify one element of the array.

Return Value

None

Remarks

Data collection started by this method without the **ACSC_DCF_SYNC** flag is called *system data collection*.

Data collection started with the **ACSC_DCF_SYNC** flag, is called *axis data collection*.

Data collection started with **ACSC_DCF_CYCLIC** flag, called *cyclic data collection*. Unlike the standard data collection that finishes when the collection array is full, cyclic data collection does not self-terminate. Cyclic data collection uses the collection array as a cyclic buffer and can continue to collect data indefinitely. When the array is full, each new sample overwrites the oldest sample in the array.

Cyclic data collection can only be terminated by calling the *StopCollect* method.

The array that stores the samples can be one or two-dimensional. A one-dimensional array is allowed only if the variable list contains one variable name.

The number of the array rows must be equal to or more than the number of variables inthe variable list. The number of the array columns must be equal to or more than the number of samples specified by the **nSample** argument.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous call example
Ch.DataCollectionExt(DataCollectionFlags.ACSC_DCF_WAIT,
    Axis.ACSC_AXIS_0, "MyArrayForDC", 100, 100,
    "FPOS(0)\rFVEL(0)\rFACC(0)");
// Asynchronous call example
ACSC_WAITBLOCK wb =
Ch.DataCollectionExtAsync(DataCollectionFlags.ACSC_DCF_WAIT,
    Axis.ACSC_AXIS_0, "MyArrayForDC", 100, 100,
```

```
                "FPOS(0)\rFVEL(0)\rFACC(0)");
            Object o = Ch.GetResult(wb, 2000);
```

3.27.2 StopCollect

Description

The method terminates data collection.

Syntax

object.StopCollect()
Async Syntax

ACSC_WAITBLOCK object.StopCollectAsync()

Arguments

None

Return Value

None

Remarks

The usual system data collection finishes when the required number of samples is collected or the **StopCollect** method is executed. The application can wait for data collection end withthe **WaitCollectEndExt** method.

The temporal data collection runs until the **StopCollect** method is executed.

The method terminates the data collection prematurely. The application can determine the number of actually collected samples from the **S_DCN** variable and the actual sampling period from the **S_DCP** variable.

If the method fails, the Error object is filled with the Error Description.

Example

```
Api acsApi = new Api();
string ArrayName = "DCA(2)(1000)";
string Vars = "FPOS(0)\rFVEL(0)\r";
acsApi.OpenCommSimulator();

acsApi.Enable(Axis.ACSC_AXIS_0);
acsApi.WaitMotorEnabled(Axis.ACSC_AXIS_0,1,5000);
acsApi.Enable(Axis.ACSC_AXIS_2);
acsApi.WaitMotorEnabled(Axis.ACSC_AXIS_2, 1, 5000);

acsApi.SetFPosition(Axis.ACSC_AXIS_0, 0);
acsApi.ExtToPoint(MotionFlags.ACSC_AMF_VELOCITY, Axis.ACSC_AXIS_0, 10000,
5000, 1000);
acsApi.DeclareVariable(AcsplVariableType.ACSC_REAL_TYPE, ArrayName);
acsApi.DataCollectionExt(DataCollectionFlags.ACSC_NONE, Axis.ACSC_AXIS_0,
ArrayName, 1000, 1, Vars);
Thread.Sleep(TimeSpan.FromSeconds(0.5));
```

```
acsApi.StopCollect();

string command = "?AST(0).3";

string reply = acsApi.Transaction(command).Trim();

if (reply == "0")
{
    Console.WriteLine("StopCollect Funcation PASS");
}
else
{
    Console.WriteLine("StopCollect Funcation FAIL");
}


acsApi.CloseComm();
```

Async Example

```
private void StopCollect()
 {
     ACSC_WAITBLOCK waitObject;
     string ArrayName = "DCA(2)(1000)";
     string res;
     string Vars = "FPOS(" + (int)GlobalAxis + "), RPOS(" +
(int)GlobalAxis + ")";
     bool success = true;

     try
     {
         AsyncTxtFile.WriteLine();
         AsyncTxtFile.WriteLine("* * * * * * StopCollect() * * * * * *
");
         AsyncTxtFile.WriteLine();
         AsyncTxtFile.WriteLine("    StopCollect");
         AsyncTxtFile.WriteLine();

         // matrix consisting of two rows with 1000 columns each
         // positions of Axes will be collected

         acsApi.EnableAsync(GlobalAxis);
         acsApi.WaitMotorEnabled(GlobalAxis, 1, 5000);
         acsApi.SetFPositionAsync(GlobalAxis, 0);
         acsApi.ExtToPointAsync(MotionFlags.ACSC_AMF_VELOCITY |
MotionFlags.ACSC_AMF_ENDVELOCITY, GlobalAxis, 10000, 5000, 1000);
         acsApi.DeclareVariableAsync(AcsplVariableType.ACSC_REAL_TYPE,
ArrayName);
```

```
          acsApi.CollectBAsync(DataCollectionFlags.ACSC_NONE, ArrayName,
1000, 1, Vars);
          System.Threading.Thread.Sleep(500);

          if (is_Async.Equals(true))
          {
              waitObject = acsApi.StopCollectAsync();
              acsApi.GetResult(waitObject, 2000);
          }
          else acsApi.StopCollect();

          res = acsApi.Transaction("?AST" + (int)GlobalAxis + ".3");
          Thread.Sleep(1500);
          if (Convert.ToDouble(res.Trim()).Equals(1))
          {
              AsyncTxtFile.WriteLine("    FAILED: DC is not stpeed.");
              if (FailedFunctions.Contains("StopCollect").Equals(false))
                  FailedFunctions.Add("StopCollect");
              success = false;
          }

          if (success.Equals(true))
              AsyncTxtFile.WriteLine("    PASSED");
      }

      catch (ACSException Ex)
      {
          AsyncTxtFile.WriteLine("    Function failed.  Exception:  " +
Ex.Message);
          AsyncTxtFile.Flush();
          if (FailedFunctions.Contains("StopCollect").Equals(false))
              FailedFunctions.Add("StopCollect");
      }
  }
```

3.27.3 SPDataCollectionStart

Description

SPDataCollectionStart(Servo Processor Data Collection) performs fast data collection and accumulates data about the specified Servo Processor variable with a constant maximum sampling rate of 20kHz. A typical use for SPDataCollectionStartis for collecting position error (**PE**) and feedback position (**FPOS**) data at the fast Servo Processor rate.

The Servo Processor value is different from the MPU value. The Servo Processor always uses counts and not units. The Servo Processor position value is not affected by a SET FPOS command. An offset is added at the MPU level only. The formula (available in the manuals) is:

$$FPOS = FP*EFAC + EOFFS$$

where **FPOS** is the MPU variable and **FP** is the Servo Processor calculated value.

SP data collection may terminate due to:

> Calling **SPDataCollectionStop**

> Accumulation of the defined number of samples

Syntax

object. SPDataCollectionStart ([ServoProcessorDataCollectionFlags flags], string array, int nSample, double period, ServoProcessor servoProcessorIndex, int servoProcessorAddress1, int servoProcessorAddress2, int servoProcessorAddress3, int servoProcessorAddress4)

Async Syntax

ACSC_WAITBLOCK object. SPDataCollectionStartAsync([ServoProcessorDataCollectionFlags flags], string array, int nSample, double period, ServoProcessor servoProcessorIndex, int servoProcessorAddress1, int servoProcessorAddress2, int servoProcessorAddress3, int servoProcessorAddress4)

Arguments

Flags	ACSC_SPDCF_REAL_TYPE - defines the array as real.
Array	Array name, up to **XARRSIZE** variable value. Array should be declared as STATIC. By default, Array is assumed to be an integer array, if the ACSC_SPDCF_REAL_TYPE switch is added, it defines the array as real.
NSample	The number of samples to collect, the maximum value depends on the size of the array.
Period	The time, in milliseconds, that each sample is taken.
ServoProcessorIndex	The index of the Servo Processor to be sampled
ServoProcessorAddress1	The address of the Servo Processor variable in the Servo Processor to sample.
ServoProcessorAddress2	As an option, you can add another address of another Servo Processor variable in the Servo Processor to sample. In this case, the array should be defined as (2)(N)
ServoProcessorAddress3	As an option, you can add another address of another Servo Processor variable in the Servo Processor to sample, In this case, the array should be defined as (3)(N).
ServoProcessorAddress4	As an option, you can add another address of another Servo Processor variable in the Servo Processor to sample. In this case, the array should be defined as (4)(N)

Return Value

None

Comments

Only one SPDC command per Servo Processor can run at a given time.

Variable	Axis	Servo Processor Variable
Position Error	0,1,2,3	axes[0].PE
Feedback Position	0,1,2,3	axes[0].fpos
Feedback Velocity	0,1,2,3	axes[0].fvel
Sin Analog Input	0,1,2,3	axes[0].sin
Cos Analog Input	0,1,2,3	axes[0].cos
Phase A Current	0,1,2,3	axes[0].is
Phase B Current	0,1,2,3	axes[0].it
Current Command	0,1,2,3	axes[0].command

Example

```
string dataArray = "MyArray(2)(10000)";
int samples = 10000;
double period = 0.5;
string SP_Varaible = "axes[0].fpos"; // Servo Processor variable name
double FPOSAddress; // SP variable address.
FPOSAddress = api.GetSPAddress(
EtherCATFlags.ACSC_ETHERCAT_NETWORK_0, // EtherCAT
ServoProcessor.ACSC_SP_0, // SP Index
SP_Varaible); // SP variable name.
string SP_Variable2 = "axes[0].cos"; // Servo Processor variable name
double COSAddress;
COSAddress = api.GetSPAddress(
EtherCATFlags.ACSC_ETHERCAT_NETWORK_0, // EtherCAT
ServoProcessor.ACSC_SP_0, // SP Index
SP_Variable2); // SP variable name.
// Declaring static array
api.DeclareVariable(AcsplVariableType.ACSC_STATIC_REAL_TYPE, dataArray);
api.SPDataCollectionStart(
ServoProcessorDataCollectionFlags.ACSC_SPDCF_REAL_TYPE,
// SP Flags
"MyArray", // Static array type real
samples,   // Number of samples
period,    // Period time in milliseconds
ServoProcessor.ACSC_SP_0, // SP Index
(int)FPOSAddress, // SP Address
(int)COSAddress,  // SP Address
```

```
-1,  // SP Address
-1); // SP Address
```

3.27.4 SPDataCollectionStop

Description

The **SPDataCollectionStop** function Immediately terminates Servo Processor data collection started by **SPDataCollectionStart** for the specified servo processor.

Syntax

object.SPDataCollectionStop([ServoProcessor servoProcessorIndex])

Async Syntax

ACSC_WAITBLOCK object.SPDataCollectionStop([ServoProcessor servoProcessorIndex])

Arguments

ServoProcessorIndex	The index of the Servo Processor.

Return Value

None

Example

```
double FPOSAddress = acsApi.GetSPAddress(
EtherCATFlags.ACSC_ETHERCAT_NETWORK_0, // First EtherCAT Network
                       ServoProcessor.ACSC_SP_0, // SP Index
                       "axes[0].fpos"); // SP variable name.


acsApi.SPDataCollectionStart(
                ServoProcessorDataCollectionFlags.ACSC_SPDCF_REAL_TYPE,
// SP Flags
                "MyArray", // Static array type real
                10000, // Number of samples
                0.5, // Period time in milliseconds
                ServoProcessor.ACSC_SP_0, // SP Index
                (int)FPOSAddress, // SP Address
                -1, // SP Address
                -1, // SP Address
                -1); // SP Address

            acsApi.WaitSPDataCollectionEnd(10000, ServoProcessor.ACSC_SP_
0);

            acsApi.SPDataCollectionStop(ServoProcessor.ACSC_SP_0);
```

3.27.5 GetSPAddress

Description

GetSPAddress reads a value from the specified SP address.

Syntax

Object.GetSPAddress ([EtherCATFlags flags], [ServoProcessor servoProcessorIndex], string servoProcessorVariable)

Async Syntax

ACSC_WAITBLOCK Object.GetSPAddress ([EtherCATFlags flags], [ServoProcessor servoProcessorIndex], string servoProcessorVariable)

Arguments

Flags	Bit-mapped argument that can include one or more of the following flags: ACSC_ETHERCAT_NETWORK_0 - first network specifier ACSC_ETHERCAT_NETWORK_1 - second network specifier ACSC_ETHERCAT_NETWORK_0 and ACSC_ETHERCAT_NETWORK_1 flags must not be specified together 0, //No flags are set (equivalent to flag ACSC_ETHERCAT_NETWORK_0)
ServoProcessorIndex	The SP index in the system.
ServoProcessorVariable	String representing the name of an SP variable.

Return Value

double - value read from SP address

Example

```
string SP_Variable = "axes[0].fpos"; // Servo Processor variable name
int FPOSAddress; // SP variable address.
FPOSAddress = acsApi.GetSPAddress(
EtherCATFlags.ACSC_ETHERCAT_NETWORK_0, // EtherCAT Network Flag 0 = First
Network
ServoProcessor.ACSC_SP_0, // SP Index
SP_Variable); // SP variable name.
```

3.27.6 WaitSPDataCollectionEnd

Description

The function waits for the end of Servo Processor data collection.

Syntax

Object.WaitSPDataCollectionEnd(int timeout, [ServoProcessor servoProcessorIndex])

Arguments

Timeout	Maximum wait time in milliseconds. If INFINITE - timeout interval never elapses
ServoProcessorIndex	The index of the Servo Processor.

Return Value

None

Comments

The function does not return while data collection is in progress and the correct parameters have been passed.

Example

```
string dataArray = "MyArray(2)(10000)";
int samples = 10000;
double period = 0.5;
int timeout = 10000;

string SP_Varaible = "axes[0].fpos"; // Servo Proccesor variable name
double FPOSAddress; // SP variable address.
FPOSAddress = acsApi.GetSPAddress(
                EtherCATFlags.ACSC_ETHERCAT_NETWORK_0, // EtherCAT
Netowrk Flag 0 = First Network
                ServoProcessor.ACSC_SP_0, // SP Index
                SP_Varaible); // SP variable name.

            string SP_Variable2 = "axes[0].cos"; // Servo Proccesor
variable name
            double COSAddress;
            COSAddress = acsApi.GetSPAddress(
                EtherCATFlags.ACSC_ETHERCAT_NETWORK_0, // EtherCAT
Netowrk Flag 0 = First Network
                ServoProcessor.ACSC_SP_0, // SP Index
                SP_Variable2); // SP variable name.


            // Declaring static array
            acsApi.DeclareVariable(AcsplVariableType.ACSC_STATIC_REAL_
TYPE, dataArray);

            acsApi.SPDataCollectionStart(
                ServoProcessorDataCollectionFlags.ACSC_SPDCF_REAL_TYPE,
// SP Flags
                "MyArray", // Static array type real
                samples, // Number of samples
                period, // Period time in milliseconds
                ServoProcessor.ACSC_SP_0, // SP Index
```

```
                  (int)FPOSAddress, // SP Address
                  (int)COSAddress, // SP Address
                  -1, // SP Address
                  -1); // SP Address

            acsApi.WaitSPDataCollectionEnd(timeout, ServoProcessor.ACSC_
SP_0);

            acsApi.SPDataCollectionStop(ServoProcessor.ACSC_SP_0);
```

3.28 Status Report Methods

The Status Report methods are:

Table 3-30. Status Report Methods

Method	Description
GetMotorState	Retrieves the current motor state.
GetAxisState	Retrieves the current axis state.
GetIndexState	Retrieves the current state of the index and mark variables.
ResetIndexState	Resets the specified bit of the index/mark state.
GetProgramState	Retrieves the current state of the program buffer.

3.28.1 GetMotorState

Description

The method retrieves the current motor state.

Syntax

object.GetMotorState(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GetMotorStateAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Remarks

The retrieved value is a 32-bit word and may include one or more of the following flags:

ACSC_MST_ENABLE - a motor is enabled

ACSC_MST_INPOS - a motor has reached a target position

ACSC_MST_MOVE - a motor is moving

ACSC_MST_ACC - a motor is accelerating

The retrieved value consists of bits held in the MST array. These are the only flags retrieved by the function, other flags held in the **MST** array are not returned.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous get motor state
MotorStates state = Ch.GetMotorState(Axis.ACSC_AXIS_0);

// Asynchronous get motor state
ACSC_WAITBLOCK wb = Ch.GetMotorStateAsync(Axis.ACSC_AXIS_0);
MotorStates state1 = (MotorStates)Ch.GetResult(wb, 2000);
```

3.28.2 GetAxisState

Description

The method retrieves the current axis state.

Syntax

object.GetAxisState(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GetAxisStateAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

AxisStates

Remarks

The *Return Value* can comprise one or more flags. For *Example*, **ACSC_AST_LEAD** (leading axis), **ACSC_AST_DC** (data collection in progress for the axis), etc. For the complete list of flags, see Axis State Flags.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous get axis state
AxisStates state = Ch.GetAxisState(Axis.ACSC_AXIS_0);

// Asynchronous get axis state
ACSC_WAITBLOCK wb = Ch.GetAxisStateAsync(Axis.ACSC_AXIS_0);
AxisStates state1 = (AxisStates)Ch.GetResult(wb, 2000);
```

3.28.3 GetIndexState

Description

The method retrieves the current set of bits that indicate the index and mark state.

Syntax

object.GetIndexState(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GetIndexStateAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

IndexStates

Remarks

The *Return Value* can include one or more of the following flags:

ACSC_IST_IND – a primary encoder index of the specified axis is latched

ACSC_IST_IND2 – a secondary encoder index of the specified axis is latched

ACSC_IST_MARK – a MARK1 signal has been generated and position of the specified axis was latched

ACSC_IST_MARK2 – a MARK2 signal has been generated and position of the specified axis was latched

The controller processes index/mark signals as follows:

When an index/mark signal is encountered for the first time, the controller latches feedback positions and raises the corresponding bit. As long as a bit is raised, the controller does not latch feedback position even if the signal occurs again. Toresume latching logic, the application must call ResetIndexState to explicitly reset the corresponding bit.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous get index state
IndexStates state = Ch.GetIndexState(Axis.ACSC_AXIS_0);

// Asynchronous get index state
ACSC_WAITBLOCK wb = Ch.GetIndexStateAsync(Axis.ACSC_AXIS_0);
IndexStates state1 = (IndexStates)Ch.GetResult(wb, 2000);
```

3.28.4 ResetIndexState

Description

The method resets the specified bit of the index/mark state.

Syntax

object.ResetIndexState(Axis axis, IndexStates mask)

Async Syntax

ACSC_WAITBLOCK object.ResetIndexStateAsync(Axis axis, IndexStates mask)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
mask	The parameter contains bit to be cleared. Only one of the following flags can be specified: **ACSC_IST_IND** – a primary encoder index of the specified axis is latched **ACSC_IST_IND2** – a secondary encoder index of the specified axis is latched **ACSC_IST_MARK** – a MARK1 signal has been generated and position of the specified axis was latched **ACSC_IST_MARK2** – a MARK2 signal has been generated and position of the specified axis was latched

Return Value

None

Remarks

The method resets the specified bit of the index/mark state. **mask** contains a bit that must be cleared, i.e., the method resets only that bit of the index/mark state, which corresponds to non- zero bit of **mask**. To get the current index/mark state, use the GetIndexState method.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Resets the specified bit of the index/mark state
Ch.ResetIndexState(Axis.ACSC_AXIS_0, IndexStates.ACSC_IST_IND);
Ch.ResetIndexState(Axis.ACSC_AXIS_0, IndexStates.ACSC_IST_IND2);
Ch.ResetIndexState(Axis.ACSC_AXIS_0, IndexStates.ACSC_IST_MARK);
Ch.ResetIndexState(Axis.ACSC_AXIS_0, IndexStates.ACSC_IST_MARK2);
IndexStates state = Ch.GetIndexState(Axis.ACSC_AXIS_0);
```

3.28.5 GetProgramState

Description

The method retrieves the current state of the program buffer.

Syntax

object.GetProgramState(ProgramBuffer buffer)

Async Syntax

ACSC_WAITBLOCK object.GetProgramStateAsync(ProgramBuffer buffer)

Arguments

buffer	Number of the buffer in which the program resides.

Return Value

ProgramStates

Remarks

The retrieved value can include one or more of the following flags:

ACSC_PST_COMPILED – a program in the specified buffer is compiled

ACSC_PST_RUN – a program in the specified buffer is running

ACSC_PST_AUTO – an auto routine in the specified buffer is running

ACSC_PST_DEBUG – a program in the specified buffer is executed in debug mode, i.e. breakpoints are active

ACSC_PST_SUSPEND – a program in the specified buffer is suspended after the step execution or due to breakpoint in debug mode

If the method fails, the Error object is filled with the Error Description.

Example

```
// Appends 2 lines to buffer 0
Ch.AppendBuffer(ProgramBuffer.ACSC_BUFFER_0, "!Empty buffer");
Ch.AppendBuffer(ProgramBuffer.ACSC_BUFFER_0, "stop");
//Run buffer 0
Ch.RunBuffer(ProgramBuffer.ACSC_BUFFER_0, null);
// Retrieves the current state of the program buffer
ProgramStates pstate = Ch.GetProgramState(
ProgramBuffer.ACSC_BUFFER_0);
```

3.29 Inputs/Outputs Access Methods

The Inputs/Outputs Access methodsare:

Table 3-31. Inputs/Outputs Access Methods

Method	Description
GetInput	Retrieves the current state of the specified digital input.
GetInputPort	Retrieves the current state of the specified digital input port.
GetOutput	Retrieves the current state of the specified digital output.
GetOutputPort	Retrieves the current state of the specified digital output port.
SetOutput	Sets the specified digital output to the specified value.

Method	Description
SetOutputPort	Sets the specified digital output port to the specified value.
GetAnalogInputNT	Retrieves the current value of the specified analog input.
GetAnalogOutputNT	Retrieves the current value of the specified analog output.
SetAnalogOutputNT	Writes the specified value to the specified analog output.
GetExtInput	Retrieves the current state of the specified extended input.
GetExtInputPort	Retrieves the current state of the specified extended input port.
GetExtOutput	Retrieves the current state of the specified extended output.
GetExtOutputPort	Retrieves the current state of the specified extended output port.
SetExtOutput	Sets the specified extended output to the specified value.
SetExtOutputPort	Sets the specified extended output port to the specified value.

3.29.1 GetInput

Description

The method retrieves the current state of the specified digital input.

Syntax

object.GetInput(int port, int bit)

Async Syntax

ACSC_WAITBLOCK object.GetInputAsync(int port, int bit)

Arguments

port	Number of the input port.
bit	Number of the specific bit.

Return Value

Int32.

The method retrieves the current state of the specified digital input.

Remarks

To get values of all inputs of the specific port, use the *GetInputPort* method.

Digital inputs are represented in the controller variable **IN**. For more information about digital inputs, see the *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 2000;
// Synchronous call example
// The method reads input 0 of port 0 (IN(0).0)
int port = Ch.GetInput(0, 0);
// Asynchronous call example
ACSC_WAITBLOCK wb = Ch.GetInputAsync(0, 0);
int port1 = (int)Ch.GetResult(wb, timeout);
```

3.29.2 GetInputPort

Description

The method retrieves the current state of the specified digital input port.

Syntax

object.GetInputPort(int port)

Async Syntax

ACSC_WAITBLOCK object.GetInputPortAsync(int port)

Arguments

port	Number of the input port.

Return Value

Int32.

Remarks

To get the value of the specific input of the specific port, use the *GetInput* method.

Digital inputs are represented in the controller variable **IN**. For more information about digital inputs, see the *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
int[] port = new int[16];
// The method reads input port 0 to 15 ( IN(0) to IN(15) )
for (int index = 0; index < port.Length; index++)
{
    port[index] = Ch.GetInputPort(index);
}
```

3.29.3 GetOutput

Description

The method retrieves the current state of the specified digital output.

Syntax

object.GetOutput(int port, int bit)

Async Syntax

ACSC_WAITBLOCK object.GetOutputAsync(int port, int bit)

Arguments

port	Number of the output port.
bit	Number of the specific bit.

Return Value

Int32.

Remarks

To get values of all outputs of the specific port, use *GetExtOutputPort*.

Digital outputs are represented in the controller variable **OUT**. For more information about digital outputs, see the *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous call example
// The method reads output 0 of port 0 ( OUT(0).0 )
int port = Ch.GetOutput(0, 0);
// Asynchronous call example
int timeout = 2000;
ACSC_WAITBLOCK wb = Ch.GetOutputAsync(0, 0);
int port1 = (int)Ch.GetResult(wb, timeout);
```

3.29.4 GetOutputPort

Description

The method retrieves the current state (0 or 1) of the specified digital output port.

Syntax

object.GetOutputPort(int port)

Async Syntax

ACSC_WAITBLOCK object.GetOutputPortAsync(int port)

Arguments

port	Number of the output port.

Return Value

Int32.

Remarks

To get the value of the specific output of the specific port, use *GetOutput*.

Digital outputs are represented in the controller variable **OUT**. For more information about digital outputs, see *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
int[] port = new int[16];
// The method reads output port 0 to 15 ( OUT(0) to OUT(15) )
for (int index = 0; index < port.Length; index++)
{
    port[index] = Ch.GetOutputPort(index);
}
```

3.29.5 SetOutput

Description

The method sets the specified digital output to the specified value.

Syntax

object.SetOutput(int port, int bit, int value)

Async Syntax

ACSC_WAITBLOCK object.SetOutputAsync(int port, int bit, int value)

Arguments

port	Number of the output port.
bit	Number of the specific bit.
value	The value to be writes to the specified output. Any non-zero value is interpreted as 1.

Return Value

None

Remarks

The method sets the specified digital output to the specified value. To set values of all outputs of a specific port, use the *SetOutputPort* method.

Digital outputs are represented in the controller variable **OUT**. For more information about digital outputs, see the *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous call example
// The method sets output 0 of port 0 to 1
// ACSPL+ equivalent: OUT(0).0 = 1
Ch.SetOutput(0, 0, 1);
```

3.29.6 SetOutputPort

Description

The method sets the specified digital output port to the specified value.

Syntax

object.SetOutputPort(int port, int value)

Async Syntax

ACSC_WAITBLOCK object.SetOutputPortAsync(int port, int value)

Arguments

port	Number of the digital output port.
value	The value to be writen to the specified port.

Return Value

None

Remarks

The method sets the specified digital output port to the specified value. To set the value of the specific output of the specific port, use the *SetOutput* method.

Digital outputs are represented in the controller variable **OUT**. For more information about digital outputs, see the *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// The method sets outputs 0 to 15 to 100
for (int index = 0; index < 16; index++)
    Ch.SetOutputPort(index, 100);
```

3.29.7 GetAnalogInputNT

Description

The function retrieves the current value of the specified analog input.

Syntax

object.GetAnalogInputNT(int port)

Async Syntax

ACSC_WAITBLOCK object.GetAnalogInputNTAsync(int port)

Arguments

port	Index of analog input port - a number between 0 and up to the maximum number of analog input signals minus one.

Return Value

Double

Variable that receives the current value of a specific analog input. Units: in scaling, by percent, of the signal and ranges from -100 to +100 of the maximum level.

Remarks

The function retrieves the current value of specified analoginput.

Analog inputs are represented in the controller variable **AIN**. For more information about analog inputs, see *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous call example
double value = Ch.GetAnalogInputNT(0);
// Asynchronous call example
int timeout = 2000;
ACSC_WAITBLOCK wb = Ch.GetAnalogInputNTAsync(0);
double value1 = (double)Ch.GetResult(wb, timeout);
```

3.29.8 GetAnalogOutputNT

Description

The function retrieves the current value of the specified analog output.

Syntax

object.GetAnalogOutputNT(int port)

Async Syntax

ACSC_WAITBLOCK object.GetAnalogOutputNTAsync(int port)

Arguments

port	Index of analog output port - a number between 0 and up to the maximum number of analog output signals minus one.

Return Value

Double

Variable that receives the current value of a specific analog output. Units: in scaling, by percent, of the signal and ranges from -100 to +100 of the maximum level.

Remarks

The method retrieves the current value of the specified analog output. To write a value to the specific analog outputs use the SetAnalogOutputNTmethod.

Analog outputs are represented in the controller variable **AOUT**. For more information about analog outputs, see *SPiiPlus ACSPL+ Programmer's Guide*.

Example

```
// Synchronous call example
double value = Ch.GetAnalogOutputNT(0);
// Asynchronous call example
int timeout = 2000;
ACSC_WAITBLOCK wb = Ch.GetAnalogOutputNTAsync(0);
double value1 =(double)Ch.GetResult(wb, timeout);
```

3.29.9 SetAnalogOutputNT

Description

The function writes the specified value to the specified analog output.

Syntax

object.SetAnalogOutputNT(int port, double value)

Async Syntax

ACSC_WAITBLOCK object.SetAnalogOutputNTAsync(int port, double value)

Arguments

port	Index of analog output port - a number between 0 and up to the maximum number of analog output signals minus one.
value	The value to be written to the specified analog output. Units: in scaling, by percent, of the signal and ranges from -100 to +100 of the maximum level.

Return Value

None

Remarks

The function writes the specified value to the specified analog outputs. To get a value of the specific analog output, use the GetAnalogOutputNT method.

Analog outputs are represented in the controller variable **AOUT**. For more information about analog outputs, see the *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// The method writes the value 100 to the analog outputs of port outputs
// 0 to 15
for (int index = 0; index < 16; index++)
{
  Ch.SetAnalogOutputNT(index,100);
}
```

3.29.10 GetExtInput

Description

The method retrieves the current state of the specified extendedinput.

Syntax

object.GetExtInput(int port, int bit)

Async Syntax

ACSC_WAITBLOCK object.GetExtInputAsync(int port, int bit)

Arguments

port	Number of the extended input port.
bit	Number of the specific bit.

Return Value

Int32.

The method retrieves the current state (0 or 1) of the extended input specified by **port** and **bit**.

Remarks

To get that states for all the inputs of the specific extended port, use the *GetExtInputPort* method.

Extended inputs are represented in the controller variable **EXTIN**. For more information about extended inputs, see *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
            // Synchronous call example
            // The method reads extended input 0 of port 0 ( EXTIN(0).0 )
            int port = Ch.GetExtInput(0, 0);

            // Asynchronous call example
            int timeout = 2000;
            ACSC_WAITBLOCK wb = Ch.GetExtInput(0,0);
           int port1 = (int)Ch.GetResult(wb, timeout);
```

3.29.11 GetExtInputPort

Description

The method retrieves the current state of the specified extended input port.

Syntax

object.GetExtInputPort(int port)

Async Syntax

ACSC_WAITBLOCK object.GetExtInputPortAsync(int port)

Arguments

port	Number of the extended input port.

Return Value

Int32.

Remarks

To get the value of a specific input of the specific extended port, use the *GetExtInput* method.

Extended inputs are represented in the controller variable **EXTIN**. For more information about extended inputs, see *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

Synchronous call example

```
// The method reads extended input 0 of port 0 ( EXTIN(0).0 )
int[] port = new int[16];
ACSC_WAITBLOCK wb = api.GetExtInputPortAsync(0 + 0);
int timeout = 2000;
```

Asynchronous call example

```
int port1 = (int)api.GetResult(wb, timeout);
for (int index = 0; index < port.Length; index++)
{
port[index] = api.GetExtInputPort(index);
}
/*
}
// The method reads input extended input port 0 to 15

// ( EXTIN(0) to EXTIN(15) )
}
*/
```

3.29.12 GetExtOutput

Description

The method retrieves the current state of the specified extended output.

Syntax

object.GetExtOutput(int port, int bit)

Async Syntax

ACSC_WAITBLOCK object.GetExtOutputAsync(int port, int bit)

Arguments

port	Number of the extended output port.
bit	Number of the specific bit.

Return Value

Int32.

The method retrieves the current state (0 or 1) of the specified extended output.

Remarks

To get values of all outputs of the specific extended port, use the *GetExtOutputPort* method.

Extended outputs are represented in the controller variable **EXTOUT**. For more information about extended outputs, see *SPiiPlus ACSPL+ Programmer's Guide.*

If the method fails, the Error object is filled with the Error Description.

Example

```
// Synchronous call example
// The method reads extended output 0 of port 0 ( EXTOUT(0).0 )

int port = Ch.GetExtOutput(0,0);
```

3.29.13 GetExtOutputPort

Description

The method retrieves the current state of the specified extended output port.

Syntax

object.GetExtOutputPort(int port)

Async Syntax

ACSC_WAITBLOCK object.GetExtOutputPortAsync(int port)

Arguments

port	Number of the extended output port.

Return Value

Int32.

The method retrieves the current state (0 or 1) of the specified extended output port.

Remarks

To get the value of the specific output of the specific extended port, use the *GetExtOutput* method.

Extended outputs are represented in the controller variable **EXTOUT**. For more information about extended outputs, see *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
int[] port = new int[16];
// The method reads input extended output port 0 to 15
// ( EXTOUT(0) to EXTOUT(15) )
for (int index = 0; index < port.Length; index++)
{
    port[index] = Ch.GetExtOutputPort(index);
}
```

3.29.14 SetExtOutput

Description

The method sets the specified extended output to the specified value.

Syntax

object.SetExtOutput(int port, int bit, int value)

Async Syntax

ACSC_WAITBLOCK object.SetExtOutputAsync(int port, int bit, int value)

Arguments

port	Number of the extended output port.
bit	Number of the specific bit.
value	The value to be written to the specified output. Any non-zero value is interpreted as 1.

Return Value

None

Remarks

The method sets the specified extended output to the specified value. To set values of all outputs of the specific extended port, use the *SetExtOutputPort* method.

Extended outputs are represented in the controller **EXTOUT** variable. For more information about extended outputs, see the *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// The method sets output 0 of extended port 0 to 1
// ACSPL+ equivalent: EXTOUT(0).0 = 1

Ch.SetExtOutput(0,0,1);
```

3.29.15 SetExtOutputPort

Description

The method sets the specified extended output port to the specified value.

Syntax

object.SetExtOutputPort(int port, int value)

Async Syntax

ACSC_WAITBLOCK object.SetExtOutputPortAsync(int port, int value)

Arguments

port	Number of the extended output port.
value	The value to be written to the specified output port.

Return Value

None

Remarks

The method sets the specified extended output port to the specified value. To set the value of the specific output of the specific extended port, use the *SetExtOutput* method.

Extended outputs are represented in the controller variable **EXTOUT**. For more information about extended outputs, see the *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// The method sets outputs of extended port 0 to 15 to 100
for (int index = 0; index < 16; index++)
{
  Ch.SetExtOutputPort(index, 100);
}
```

3.30 Safety Control Methods

The Safety Control methods are:

Table 3-32. Safety Control Methods

Method	Description
GetFault	Retrieves the set of bits that indicate the motor or system faults.
SetFaultMask	Sets the mask that enables/disables the examination and processing of the controller faults.
GetFaultMask	Retrieves the mask that defines which controller faults are examined and processed.
EnableFault	Enables the specified axis or system fault.
DisableFault	Disables the specified axis or system fault.
SetResponseMask	Sets the mask that defines for which axis or system faults the controller provides default response.
GetResponseMask	Retrieves the mask that defines for which axis or system faults the controller provides default response.
EnableResponse	Enables the default response to the specified axis or system fault.
DisableResponse	Disables the default response to the specified axis or system fault.
GetSafetyInput	Retrieves the current state of the specified safety input.
GetSafetyInputPort	Retrieves the current state of the specified safety input port.
GetSafetyInputPortInv	Retrieves the set of bits that define inversion for the specified safety input port.
SetSafetyInputPortInv	Sets the set of bits that define inversion for the specified safety input port.
FaultClear	The method clears the current faults and results of previous faults stored in the MERR variable.
FaultClearM	The method clears the current faults and results of previous faults stored in the MERR variable for multiple axis.

3.30.1 GetFault

Description

The method retrieves the set of bits that indicate the motor or system faults.

Syntax

object.GetFault(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GetFaultAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

SafetyControlMasks.

The method retrieves the set of bits that indicate motor or system faults.

Remarks

The motor faults are related to a specific motor, the power amplifier, and the Servo processor. For example: Position Error, Encoder Error, or Driver Alarm.

The system faults are not related to any specific motor. For *Example*: Emergency Stop or Memory Fault.

The parameter **fault** receives the set of bits that indicates the controller faults. To recognize the specific fault, properties ACSC_SAFETY_*** can be used. See Safety Control Masks for a detailed description of these properties.

For more information about the controller faults, see the *SPiiPlus ACSPL+ Programmer's Guide*. If the method fails, the Error object is filled with the Error Description.

Example

```
// Retrieves the set of bits that indicate the motor or
// system faults
SafetyControlMasks fault = Ch.GetFault(Axis.ACSC_AXIS_0);
```

3.30.2 SetFaultMask

Description

The method sets the mask that enables or disables the examination and processing of the controller faults.

Certain controller faults provide protection against potential serious bodily injury and damage to the equipment. Be aware of the implications before disabling any alarm, limit, or error.

Syntax

object.SetFaultMask(Axis axis, SafetyControlMasks mask)

Async Syntax

ACSC_WAITBLOCK object.SetFaultMaskAsync(Axis axis, SafetyControlMasks mask)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
mask	The mask to be set: If a bit of the **mask** is zero, the corresponding fault is disabled. To set/reset a specified bit, use ACSC_SAFETY_*** properties. See Safety Control Masks for a detailed *Description* of these properties. If the **mask** is ACSC_ALL, then all the faults for the specified axis are enabled. If the **mask** is ACSC_NONE, then all the faults for the specified axis are disabled.

Return Value

None

Remarks

The method sets the mask that enables/disables the examination and processing ofthe controller faults. The two types of controller faults are motor faults and system faults.

The motor faults are related to a specific motor, the power amplifier or the Servo processor. For example: Position Error, Encoder Error or Driver Alarm.

The system faults are not related to any specific motor. For example: Emergency Stop or Memory Fault.

For more information about the controller faults, see the *SPiiPlus ACSPL+ Programmer's Guide*. If the method fails, the Error object is filled with the Error Description.

Example

```
// Enable all faults of axis 0
Ch.SetFaultMask(Axis.ACSC_AXIS_0, SafetyControlMasks.ACSC_ALL);
```

3.30.3 GetFaultMask

Description

The method retrieves the mask that defines which controller faults are examined and processed.

Syntax

object.GetFaultMask(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GetFaultMaskAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

SafetyControlMasks.

The method retrieves the mask that defines which controller faults are examined and processed.

Remarks

If a bit of the mask is zero, the corresponding fault is disabled.

Use the ACSC_SAFETY_*** properties to examine a specific bit. See Safety Control Masks for a detailed description of these properties.

Controller faults are of two types: motor faults and system faults.

The motor faults are related to a specific motor, the power amplifier or the Servo processor. For example: Position Error, Encoder Error or Driver Alarm.

The system faults are not related to any specific motor, for example: Emergency Stop or Memory Fault.

For more information about the controller faults, see the *SPiiPlus ACSPL+ Programmer's Guide*. If the method fails, the Error object is filled with the Error Description.

Example

```
// Retrieves the mask that defines which controller Faults are examined
// and processed of axis 0
SafetyControlMasks mask= Ch.GetFaultMask(Axis.ACSC_AXIS_0);
```

3.30.4 EnableFault

Description>

The method enables the specified axis or system fault.

Syntax

object.EnableFault(Axis axis, SafetyControlMasks fault)

Async Syntax

ACSC_WAITBLOCK object.EnableFaultAsync(Axis axis, SafetyControlMasks fault)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
fault	The fault to be enabled. Only one fault can be enabled at atime. To specify the fault, one of the properties ACSC_SAFETY_*** can be used. See Safety Control Masks for a detailed description of these properties.

Return Value

None

Remarks

The method enables the examination and processing of the specified motor or system fault by setting the specified bit of the fault mask to one.

The motor faults are related to a specific motor, the power amplifier, and the Servo processor. For example: Position Error, Encoder Error, and Driver Alarm.

The system faults are not related to any specific motor. For example: Emergency Stop, Memory Fault. For more information about the controller faults, see *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Enable fault Right Limit of axis 0
Ch.EnableFault(Axis.ACSC_AXIS_0,SafetyControlMasks.ACSC_SAFETY_RL);
```

3.30.5 DisableFault

Description

The method disables the specified axis or system fault.

⚠ WARNING	Certain controller faults provide protection against potential serious bodily injury and damage to equipment. Be aware of the implications before disabling any alarm, limit, or error.

Syntax

object.DisableFault(Axis axis, SafetyControlMasks fault)

Async Syntax

ACSC_WAITBLOCK object.DisableFaultAsync(Axis axis, SafetyControlMasks fault)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
fault	The fault to be disabled. Only one fault can be enabled at a time. To specify the fault, one of the properties ACSC_SAFETY_*** can be used. See Safety Control Masks for a detailed description of these properties.

Return Value

None

Remarks

The method disables the examination and processing of the specified motor or system fault by setting the specified bit of the fault mask to zero.

The motor faults are related to a specific motor, the power amplifier, and the Servo processor. For example: Position Error, Encoder Error, and Driver Alarm.

The system faults are not related to any specific motor, for example: Emergency Stop, Memory Fault. For more information about the controller faults, see *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Disable fault Software Right Limit in axis 0
api.DisableFault(Axis.ACSC_AXIS_0,SafetyControlMasks.ACSC_SAFETy_RL);
```

3.30.6 SetResponseMask

Description

The method retrieves the mask that defines the motor or the system faults for which the controller provides the default response.

> Certain controller faults provide protection against potential serious bodily injury and damage to the equipment. Be aware of the implications before disabling any alarm, limit, or error.

Syntax

object.SetResponseMask(Axis axis, SafetyControlMasks mask)

Async Syntax

ACSC_WAITBLOCK object.SetResponseMaskAsync(Axis axis, SafetyControlMasks mask)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
mask	The mask to be set: If a bit of the **mask** is zero, the corresponding fault is disabled. To set/reset a specified bit, use ACSC_SAFETY_*** properties. See Safety Control Masks for a detailed description of these properties. If the **mask** is ACSC_ALL, then all the faults for the specified axis are enabled.If the mask is ACSC_NONE, then all the faults for the specified axis are disabled.

Return Value

None

Remarks

The method retrieves the mask that defines the motor or the system faults for which the controller provides the default response.

The default response is a controller-predefined action for the corresponding fault. For more information about the controller faults and default responses, see the *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
            // Enable all default responses
            Ch.SetResponseMask(Axis.ACSC_AXIS_0,
                SafetyControlMasks.ACSC_SAFETY_AL);
```

3.30.7 GetResponseMask

Description

The method retrieves the mask that defines the motor or the system faults for which the controller provides the defaultresponse.

Syntax

object.GetResponseMask(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GetResponseMaskAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

SafetyControlMasks.

Remarks

The method retrieves the mask that determines whether the controller will respond to a motor or system fault with the fault's default response.

For more information about the controller faults and default, responses see the *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Retrieves the mask of axis 0

SafetyControlMasks mask = Ch.GetResponseMask(Axis.ACSC_AXIS_0);
```

3.30.8 EnableResponse

Description

The method enables the response to the specified axis or systemfault.

Syntax

object.EnableResponse(Axis axis, SafetyControlMasks response)

Async Syntax

ACSC_WAITBLOCK object.EnableResponseAsync(Axis axis, SafetyControlMasks response)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
response	The default response to be enabled. Only one default response can be enabled at a time. To specify the default response, one of the properties ACSC_SAFETY_*** can be used. See Safety Control Masks for a detailed description of these properties.

Return Value

None

Remarks

The method enables the default response to the specified axis or system fault by setting the specified bit of the response mask to one.

The default response is a controller-predefined action for the corresponding fault. For more information about the controller faults and default responses, see *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Enable the default response to the Position Error fault of axis 0

Ch.EnableResponse(Axis.ACSC_AXIS_0,SafetyControlMasks.ACSC_SAFETY_PE);
```

3.30.9 DisableResponse

Description

The method disables the default response to the specified axis or system fault.

⚠WARNING	Certain controller faults provide protection against potential serious bodily injury and damage to equipment. Be aware of the implications before disabling any alarm, limit, or error.

Syntax

object.DisableResponse(Axis axis, SafetyControlMasks response)

Async Syntax

ACSC_WAITBLOCK object.DisableResponseAsync(Axis axis, SafetyControlMasks response)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
response	The default response to be disabled. Only one default response can be enabled at a time. To specify the default response, one of the properties ACSC_SAFETY_*** can be used. See Safety Control Masks for a detailed description of these properties.

Return Value

None

Remarks

The method disables the default response to the specified motor or system fault by setting the specified bit of the response mask to zero.

The default response is a controller-predefined action for the corresponding fault. Formore information about the controller faults and default responses, see *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Disable the default response to the Right Limit fault

Ch.DisableResponse(Axis.ACSC_AXIS_0,SafetyControlMasks.ACSC_SAFETY_RL);
```

3.30.10 GetSafetyInput

Description

The method retrieves the current state of the specified safety input.

Syntax

object.GetSafetyInput(Axis axis, SafetyControlMasks input)

Async Syntax

ACSC_WAITBLOCK object.GetSafetyInputAsync(Axis axis, SafetyControlMasks input)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
input	The specific safety input. The safety input can be one or any combination of the following: ACSC_SAFETY_RL ACSC_SAFETY_LL ACSC_SAFETY_NETWORK ACSC_SAFETY_HOT ACSC_SAFETY_DRIVE ACSC_SAFETY_ES See Safety Control Masks for a detailed description of these properties.

Return Value

SafetyControlMasks.

The method retrieves the current state of the specified safety input.

Remarks

To get values of all safety inputs of the specific axis, use *GetSafetyInputPort*.

Safety inputs are represented in the controller variables **SAFIN** and **S_SAFIN**. For more information about safety inputs, see *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// The method reads Emergency Stop system safety input
// Reads Right Limit Safety input of axis 0
SafetyControlMasks sinput = Ch.GetSafetyInput(
    Axis.ACSC_AXIS_0, SafetyControlMasks.ACSC_SAFETY_RL);
```

3.30.11 GetSafetyInputPort

Description

The method retrieves the current state of the specified safety input port.

Syntax

object.GetSafetyInputPort(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GetSafetyInputPortAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

SafetyControlMasks.

The method retrieves the current state of the specified safety input port.

To recognize a specific motor safety input, only one of the following properties can be used:

- ACSC_SAFETY_RL
- ACSC_SAFETY_LL
- ACSC_SAFETY_NETWORK
- ACSC_SAFETY_HOT
- ACSC_SAFETY_DRIVE

To recognize a specific system safety input, only the ACSC_SAFETY_ES property can be used.

See Safety Control Masks for a detailed description of these properties.

Remarks

To get the state of the specific safety input of a specific axis, use *GetSafetyInput*.

Safety inputs are represented in the controller variables **SAFIN** and **S_SAFIN**. For more information about safety inputs, see *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// The method reads safety input port of the axis 0

SafetyControlMasks sinput = Ch.GetSafetyInputPort(Axis.ACSC_AXIS_0);
```

3.30.12 GetSafetyInputPortInv

Description

The method retrieves the set of bits that define inversion for the specified safety input port.

Syntax

object.GetSafetyInputPortInv(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.GetSafetyInputPortInvAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

SafetyControlMasks.

The method retrieves the set of bits that define inversion for the specified safety input port. To recognize a specific bit, use the following properties:

- ACSC_SAFETY_RL
- ACSC_SAFETY_LL
- ACSC_SAFETY_NETWORK
- ACSC_SAFETY_HOT
- ACSC_SAFETY_DRIVE

Use the ACSC_SAFETY_ES property to recognize an inversion for the specific system safety input port.

See Safety Control Masks for a detailed description of these properties.

Remarks

To set the specific inversion for the specific safety input port, use *GetSafetyInputPortInv*.

If a bit of the retrieved set is zero, the corresponding signal is not inverted and therefore high voltage is considered an active state. If a bit is raised, the signal is inverted and low voltage is considered an active state.

Inversions of safety inputs are represented in the controller variables SAFIN and S_SAFIN. For more information about safety inputs, see SPiiPlus ACSPL+ Programmer's Guide.

Example

```
// The method retrieves the set of bits that define inversion
// for the safety input port of the axis 0
SafetyControlMasks sinput = Ch.GetSafetyInputPortInv(Axis.ACSC_AXIS_0);
```

3.30.13 SetSafetyInputPortInv

Description

The method sets the set of bits that define inversion for the specified safety input port.

Syntax

object.SetSafetyInputPortInv(Axis axis, SafetyControlMasks value)

Async Syntax

ACSC_WAITBLOCK object.SetSafetyInputPortInvAsync(Axis axis, SafetyControlMasks value)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
value	The specific inversion. To set a specific bit, use the following properties: ACSC_SAFETY_RL ACSC_SAFETY_LL ACSC_SAFETY_NETWORK ACSC_SAFETY_HOT ACSC_SAFETY_DRIVE. To set an inversion for the specific system safety input port, use only the ACSC_ SAFETY_ES property. See Safety Control Masks for a detailed description of these properties.

Return Value

None

Remarks

The method sets the bits that define inversion for the specified safety input port. To retrieve an inversion for the specific safety input port, use the *GetSafetyInputPortInv* method.

If a bit of the set is zero, the corresponding signal will not be inverted and therefore high voltage is considered an active state. If a bit is raised, the signal will be inverted and low voltage is considered an active state.

The inversions of safety inputs are represented in the controller variables SAFIN and S_SAFIN. For more information about safety inputs, see *SPiiPlus ACSPL+ Programmer's Guide*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// The method sets the inversion for safety input port of the axis 0
Ch.SetSafetyInputPortInv(Axis.ACSC_AXIS_0,(SafetyControlMasks)100);
```

3.30.14 FaultClear

Description

The method clears the current faults and the result of the previous fault stored in the **MERR** variable.

Syntax

object.FaultClear(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.FaultClearAsync(Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions

Return Value

None

Remarks

The method clears the current faults of the specified axis and the result of the previous fault stored in the **MERR** variable.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Clears the current faults and the results of the previous faults
// stored in the MERR variable of axis 0
Ch.FaultClear(Axis.ACSC_AXIS_0);
```

3.30.15 FaultClearM

Description

The method clears the current faults and results of previous faults stored in the **MERR** variable for multiple axis.

Syntax

object.FaultClearM(Axis[] axes)

Async Syntax

ACSC_WAITBLOCK object.FaultClearMAsync(Axis[] axes)

Arguments

axes	Array of axis constants. Each element specifies one involved axis: ACSC_AXIS_0 corresponds to the 0axis, ACSC_AXIS_1 to the 1axis, etc. After the last axis, one additional element must be included that contains –1 and marks the end of the array. For the axis constants see Axis Definitions.

Return Value

None

Remarks

If the reason for the fault is still active, the controller will set the fault immediately after this command is performed. If cleared fault is Encoder Error, the feedback position is reset to zero.

If the method fails, the Error object is filled with the Error Description.

Example

```
Axis[] axes = { Axis.ACSC_AXIS_0, Axis.ACSC_AXIS_1, Axis.ACSC_NONE };
// Clears the current faults and results of the previous faults
//stored in the MERR variable of axes 0 and 1

Ch.FaultClearM(axes);
```

3.31 Wait-for-Condition Methods

The Wait-for-Condition methods are:

Table 3-33. Wait-for-Condition Methods

Method	Description
WaitMotionEnd	Waits for the end of a motion.
WaitLogicalMotionEnd	Waits for the logical end of a motion.
WaitCollectEndExt	Waits for the end of data collection.
WaitProgramEnd	Waits for the program termination in the specified buffer.
WaitMotorCommutated	Waits for the specified state of the specified motor.
WaitMotorEnabled	Waits for the specified state of the specified motor.
WaitInput	Waits for the specified state of the specified digital input.

3.31.1 WaitMotionEnd

Description

The method waits for the end of a motion.

Syntax

object.WaitMotionEnd (Axis axis, int timeout)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
timeout	Maximum waiting time in milliseconds. If **timeout** is INFINITE, the method's time-out interval never elapses.

Return Value

None

Remarks

The method does not return while the specified axis is involved in a motion, the motor has not settled in the final position and the specified time-out interval has not elapsed.

The method differs from the *WaitLogicalMotionEnd* method. Examining the same motion, the *WaitMotionEnd* method will return latter. The WaitLogicalMotionEnd method returns when the generation of the motion finishes. On the other hand, the *WaitMotionEnd* method returns when the generation of the motion finishes and the motor has settled in the final position.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 10000;

// Wait for the end of motion of axis 0 during 10 sec
Ch.WaitMotionEnd(Axis.ACSC_AXIS_0,timeout);
```

3.31.2 WaitLogicalMotionEnd

Description

The method waits for the logical end of a motion.

Syntax

object.WaitLogicalMotionEnd (Axis axis, int timeout)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
timeout	Maximum waiting time in milliseconds. If **timeout** is INFINITE, the method's time-out interval never elapses. -1 - INFINITE or 0xFFFFFFFF - INFINITE

Return Value

None

Remarks

The method does not return while the specified axis is involved in a motion and the specified time-out interval has not elapsed.

The method differs from the *WaitMotionEnd* method. Examining the same motion, the *WaitMotionEnd* method will return later. The *WaitLogicalMotionEnd* method returns when the generation of the motion finishes. On the other hand, the *WaitMotionEnd* method returns when the generation of the motion finishes and the motor has settled in the final position.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 10000;

// Wait for the logical end of motion of axis 0 during 10 sec
Ch.WaitLogicalMotionEnd(Axis.ACSC_AXIS_0,timeout);
```

3.31.3 WaitCollectEndExt

DESCRIPTION

The method waits for the end of data collection.

SYNTAX

object.WaitCollectEndExt(int timeout, int Axis)

ARGUMENTS

timeout	Maximum waiting time in milliseconds. If **timeout** is INFINITE, the method's time-out interval never elapses.
axis	Axis constant of the axis to which the data collection must be synchronized: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

RETURN VALUE

None

REMARKS

The method does not return while the system data collection is in progress and the specified time-out interval has not elapsed. The method verifies the **S_ST.#DC** system flag.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 10000;
// Wait data collection ends for 10 sec
api.WaitCollectEndExt(timeout,Axis.ACSC_AXIS_0);
```

3.31.4 WaitProgramEnd

Description

The method waits for the program termination in the specified buffer.

Syntax

object.WaitProgramEnd(ProgramBuffer buffer, int timeout)

Arguments

buffer	Buffer number
timeout	Maximum waiting time in milliseconds. If **timeout** is INFINITE, the method's time-out interval never elapses.

Return Value

None

Remarks

The method does not return while the ACSPL+ program in the specified buffer is in progress and the specified time-out interval has not elapsed.

If the method fails, the Error object is filled with the Error Description.

Example

```
            int timeout = 1000;
            for (int index = 0; index < 8; index++)
            {
                // Append a line to all buffers
                Ch.AppendBuffer((ProgramBuffer)index, "WAIT 500;");
                Ch.AppendBuffer((ProgramBuffer)index, "STOP");
                // Run all buffers
                Ch.RunBuffer((ProgramBuffer)index,null);
                // Wait program ends during 1 sec
                Ch.WaitProgramEnd((ProgramBuffer)index, timeout);
        }
```

3.31.5 WaitMotorCommutated

Description

The method waits for the specified state of the specified motor.

Syntax

object.WaitMotorCommutated(Axis axis, int state, int timeout)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
state	1 - The method waits for the motor to be commutated.
timeout	Maximum waiting time in milliseconds. If **timeout** is INFINITE, the method's time-out interval never elapses.

Return Value

None

Remarks

The method does not returnwhile the specified motor is not in the desired state and the specified time- out interval has not elapsed. The method examines the **MFLAGS.#BRUSHOK** flag.

If the method fails, the Error object is populated with the Error Description

Example

```
int timeout = 5000;
// Commut axis 0
 Ch.Commut(Axis.ACSC_AXIS_0);
// Wait motor 0 commutated during 5 sec
Ch.WaitMotorCommutated(Axis.ACSC_AXIS_0,1,timeout);
```

3.31.6 WaitMotorEnabled

Description

The method waits until the specified motor is commutated.

Syntax

object.WaitMotorEnabled (Axis axis, int state, int timeout)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
state	1 – the method waits for the motor to be enabled, 0 – the method waits for the motor to be disabled.
timeout	Maximum waiting time in milliseconds. If **timeout** is INFINITE, the method's time-out interval never elapses.

Return Value

None

Remarks

The method does not returnwhile the specified motor is not in the desired state and the specified time- out interval has not elapsed. The method examines the **MST.#ENABLED** motor flag.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
// Enable axis 0
Ch.Enable(Axis.ACSC_AXIS_0);
// Wait motor 0 enabled during 5 sec
Ch.WaitMotorEnabled(Axis.ACSC_AXIS_0,1,timeout);
```

3.31.7 WaitInput

Description

The method waits for the specified state of digital input.

Syntax

object.WaitInput (int port, int bit, int state, int timeout)

Arguments

port	Number of input port: 0 corresponds to IN0, 1 – to IN1, etc.
bit	Selects one bit from the port, from 0 to 31.
state	Specifies a desired state of the input, 0 or 1.
timeout	Maximum waiting time in milliseconds. If **timeout** is INFINITE, the method's time-out interval never elapses.

Return Value

None

Remarks

The basic configuration of the SPiiPlus PCI model provides only 16 inputs. Therefore, **Port** must be 0, and **Bit** can be specified only from 0 to 15.

If the method fails, the Error object is filled with the Error Description.

Example

```
int timeout = 5000;
// Wait for IN0.0 = 1 during 5 sec
// Parameters: 0 - IN0
//             0 - IN0.0
//             1 - wait for IN0.0 = 1
//             timeout - during 5 sec

Ch.WaitInput(0, 0, 1, timeout);
```

3.32 Event and Interrupt Handling Methods

The Event and Interrupt Handling methods are:

Table 3-34. Event and Interrupt Handling Methods

Method	Description
EnableEvent	Enables event generation for the specified interrupt condition.
DisableEvent	Disables event generation for the specified interrupt condition.
SetInterruptMask	Sets the mask for the specified interrupt.
GetInterruptMask	Retrieves the mask for the specified interrupt.

3.32.1 EnableEvent

Description

The method enables event generation for the specified interrupt condition.

Syntax

object.EnableEvent(Interrupts flags)

Arguments

flags	Specifies one of the interrupts such as ACSC_INTR_PEG, ACSC_INTR_MARK1 etc. For the full list of the interrupts, see Interrupt Types.

Return Value

None

Remarks

The library generates an event when the specified Interrupt occurs. SPiiPlus NET Library has events for all types of interrupts:

PEG (AxisMasks Param), MARK1 (AxisMasks Param) etc. For full list of SPiiPlus NETLibrary events see Events.

The bit-mapped event parameter **Param** identifies which axis/buffer/input the interrupt was generated for. See Callback Interrupt Masks for a detailed description of the **Param** argument for each interrupt.

One event can be associated with each controller interrupt. All events are handled in the main application thread, therefore the application execution is stopped until the event is handled.

To disable a specific event, call the EnableEvent method with the **flags** argument equal to the specified interrupt type.

Before the PEG interrupts can be detected, the *AssignPins* method must be called.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Enable event ACSC_INTR_ACSPL_PROGRAM
Ch.EnableEvent(Interrupts.ACSC_INTR_ACSPL_PROGRAM);
// Enable event ACSC_INTR_PROGRAM_END
Ch.EnableEvent(Interrupts.ACSC_INTR_PROGRAM_END);
// Delete all lines in buffer 5
Ch.Transaction("#5D1,100000");
// Appends a line to buffer 5
Ch.AppendBuffer(ProgramBuffer.ACSC_BUFFER_5, "interrupt stop");
// Execute buffer 5
// Ch.Transaction("#5X");
```

3.32.2 DisableEvent

Description

The method disables event generation for the specified interrupt condition.

Syntax

object.DisableEvent (Interrupts flags)

Arguments

flags	Specifies one of the interrupts such as ACSC_INTR_PEG, ACSC_INTR_MARK1 etc. For the full list of the interrupts, see Interrupt Types.

Return Value

None

Remarks

The method disables event generation that was enabled with method EnableEvent

If the method fails, the Error object is filled with the Error Description.

Example

```
// Disable event PROGRAM_END
Ch.DisableEvent(Interrupts.ACSC_INTR_PROGRAM_END);
```

3.32.3 SetInterruptMask

Description

The method sets the mask for specified interrupt.

Syntax

object.SetInterruptMask(Interrupts interrupt, Uint32 mask)

Arguments

interrupt	Specifies one of the following interrupts: **ACSC_INTR_ACSPL_PROGRAM** – an ACSPL+ program has generated the interrupt by INTERRUPT command **ACSC_INTR_ACSPL_PROGRAM_EX** – an ACSPL+ program has generated the interrupt by INTERRUPTEX command **ACSC_INTR_COMM_CHANNEL_CLOSED** - a communication channel has been closed. **ACSC_INTR_COMMAND** – a line of ACSPL+ commnds has been executed in a dynamic buffer **ACSC_INTR_EMERGENCY** - an EMERGENCY STOP signal has been generated. **ACSC_INTR_ETHERCAT_ERROR** - an EtherCAT error occurred. **ACSC_INTR_LOGICAL_MOTION_END** – a logical motion has finished **ACSC_INTR_MOTION_FAILURE** – a motion has been interrupted due to a fault **ACSC_INTR_MOTION_PHASE_CHANGE** – motion profile changes the phase **ACSC_INTR_MOTION_START** – motion starts **ACSC_INTR_MOTOR_FAILURE** – a motor has been disabled due to a fault **ACSC_INTR_NEWSEGM** - an AST.#NEWSEGM bit is high. **ACSC_INTR_PHYSICAL_MOTION_END** – a physical motion has finished **ACSC_INTR_PROGRAM_END** – an ACSPL+ program has finished **ACSC_INTR_SOFTWARE_ESTOP** - the EStop button was clicked. **ACSC_INTR_SYSTEM_ERROR** - a system error occurred. **ACSC_INTR_TRIGGER** – AST.#TRIGGER bit goes high
mask	The mask to be set. If some bit = 0 then the interrupt for the corresponding axis/buffer/input does not occur – interrupt is disabled. Use ACSC_MASK_*** to set/reset a specified bit. See GetInterruptMask for a detailed description of these enums. If **Mask** is ACSC_ALL, the interrupts for all axes/buffers/inputs are enabled. If **Mask** is ACSC_NONE, the interrupts for all axes/buffers/inputs are disabled. As default all bits for each interrupts are set to one.

Return Value

None

Remarks

The method sets the bit mask for specified interrupt. To get current mask for specified interrupt, call *GetInterruptMask*.

Using a mask, you can reduce the number of generated events. The event will be generated only if the interrupt is caused by an axis/buffer/input that corresponds to non-zero bit in therelated mask.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Sets the mask for specified interrupt only foraxis 0
Ch.SetInterruptMask(Interrupts.ACSC_INTR_PROGRAM_END,
    (uint)AxisMasks.ACSC_MASK_AXIS_0);
```

3.32.4 GetInterruptMask

Description

The method retrieves the mask for specified interrupt.

Syntax

object.GetInterruptMask(Interrupts interrupt)

Arguments

Interrupt	Specifies one of the following interrupts: **ACSC_INTR_ACSPL_PROGRAM** – an ACSPL+ program has generated the interrupt by INTERRUPT command **ACSC_INTR_ACSPL_PROGRAM_EX** – an ACSPL+ program has generated the interrupt by INTERRUPTEX command **ACSC_INTR_COMM_CHANNEL_CLOSED** - a communication channel has been closed. **ACSC_INTR_COMMAND** – a line of ACSPL+ commands executed in a dynamic buffer **ACSC_INTR_EMERGENCY** - an EMERGENCY STOP signal has been generated. A**CSC_INTR_ETHERCAT_ERROR** - an EtherCAT error occurred. **ACSC_INTR_LOGICAL_MOTION_END** – a logical motion has finished **ACSC_INTR_MOTION_FAILURE** – a motion has been interrupted due to a fault **ACSC_INTR_MOTION_PHASE_CHANGE** – motion profile changes the phase **ACSC_INTR_MOTION_START** – motion starts **ACSC_INTR_MOTOR_FAILURE** – a motor has been disabled due to a fault **ACSC_INTR_NEWSEGM** - an AST.#NEWSEGM bit is high. **ACSC_INTR_PHYSICAL_MOTION_END** – a physical motion has finished **ACSC_INTR_PROGRAM_END** – an ACSPL+ program has finished **ACSC_INTR_SOFTWARE_ESTOP** - the EStop button was clicked. **ACSC_INTR_SYSTEM_ERROR** - a system error occurred. **ACSC_INTR_TRIGGER** – AST.#TRIGGER bit goes high.

Return Value

UInt32.

The method retrieves the bit mask for the specified interrupt.

Remarks

To set the mask for a specified interrupt, call *SetInterruptMas*k.

If a bit in the Return Value equals 0, the interrupt for the corresponding axis/buffer/input is disabled.

Use the ACSC_MASK_*** properties to find the values of specific bits. See Callback Interrupt Masks for a detailed description of these properties.

By default all the interrupt bits are set to one.

If the method fails, the Error object is filled with the Error Description.

Example

```
// The example shows how to get the mask for specific interrupt
// An ACSPL+ program finished
uint mask = Ch.GetInterruptMask(Interrupts.ACSC_INTR_PROGRAM_END);
```

3.33 Variables Management Methods

The Variables Management methods are:

Table 3-35. Variables Management Methods

Method	Description
DeclareVariable	Creates the persistent global variable.
ClearVariables	Deletes all persistent global variables.

3.33.1 DeclareVariable

Description

The method creates a persistent global variable.

Syntax

object.DeclareVariable (AcsplVariableType type, string name)

Async Syntax

object.DeclareVariableAsync (AcsplVariableType type, string name)

Arguments

type	Type of the variable. For an integer variable **Type** must be **ACSC_INT_TYPE**. For a real variable, **Type** must be **ACSC_REAL_TYPE**.
name	Name of the variable.

Return Value

None

Remarks

The method creates the persistent global variable specified by **name** of type specified by **type**. The variable can be used as any other ACSPL+ or global variable.

If it is necessary to declare one or two-dimensional array, **name** should also contains the dimensional size in brackets.

The lifetime of a persistent global variable is not connected with any program buffer. The persistent variable survives any change in the program buffers and can be erased only by the *ClearVariables* method.

The method waits for the controller response.

If the method fails, the Error object is filled with the Error Description.

Example

```
// The method creates the persistent global variable "MyVar"
// as integer type.
Ch.DeclareVariable(AcsplVariableType.ACSC_INT_TYPE, "MyVar");
```

3.33.2 ClearVariables

Description

Deletes all persistent global variables.

Syntax

object.ClearVariables

Arguments

None

Return Value

None

Remarks

The method deletes all persistent global variables created by the *DeclareVariable* method. The method waits for the controller response.

If the method fails, the Error object is filled with the Error Description.

Example

```
// The method creates the persistent global variable "MyVar"
// as integer type.
Ch.DeclareVariable(AcsplVariableType.ACSC_INT_TYPE,"MyVar");
// The method deletes all persistent global variables
Ch.ClearVariables();
```

3.34 Service Methods

The Service methods are:

Table 3-36. Service Methods

Method	Description
GetLogData	Retrieves the data of firmware log.
GetFirmwareVersion	Retrieves the firmware version of the controller.
GetSerialNumber	Retrieves the controller serial number.
SysInfo	Retrieves certain system information.

Method	Description
GetBuffersCount	Returns the number of available ACSPL+ programming buffers.
GetAxesCount	Returns the number of available axes.
GetDBufferIndex	Retrieves the index of the D-Buffer.
GetUMDVersion	Retrieves UMD version string

3.34.1 GetLogData

Description

The method is used to retrieve the data of firmware log.

Syntax

object.GetLogData();

Async Syntax

ACSC_WAITBLOCK object.GetLogDataAsync()

Arguments

None

Return Value

String

Example

```
// Synchronous Get log data
string logData = Ch.GetLogData();
// Asynchronous Get log data
int timeout = 2000;
ACSC_WAITBLOCK wb = Ch.GetLogDataAsync();
string logData1 = (string)Ch.GetResult(wb, timeout);
```

3.34.2 GetFirmwareVersion

Description

The method retrieves the firmware version of the controller.

Syntax

object.GetFirmwareVersion()

Arguments

None

Return Value

String

The method retrieves the controller firmware version.

Remarks

If the method fails, the Error object is filled with the Error Description.

Example

```
// Retrieves the firmware version of the controller
string firm = Ch.GetFirmwareVersion();
```

3.34.3 GetSerialNumber

Description

The method retrieves the controller serial number.

Syntax

object.GetSerialNumber()

Arguments

None

Return Value

String

The method retrieves the character string that contains the controller serial number.

Remarks

If the method fails, the Error object is filled with the Error Description.

Example

```
// Retrieves the character string that contains the controller
// serial number
string sern = Ch.GetSerialNumber();
```

3.34.4 SysInfo

Description

The method returns certain system information based on the argument that is specified (see System Information Keys).

Syntax

object.SysInfo (int key)

Async Syntax

object.SysInfoAsync (int key)

Arguments

Key	Configuration key, specifies the configured feature. Assigns value of **key** argument in the ACSPL+ **SYSINFO** function.

Return Value

Double

Receives the configuration data.

Example

```
// Gets SPiiPlus Model Number
double value = Ch.SysInfo((int)SystemInfoKey.ACSC_SYS_MODEL_KEY);
```

3.34.5 GetBuffersCount

Description

The method returns the number of available ACSPL+ programming buffers.

Syntax

object.GetBuffersCount()

Async Syntax

ACSC_WAITBLOCK object.GetBuffersCountAsync()

Arguments

None

Return Value

Double

Receives the number of available ACSPL+ programming buffers.

Example

```
// Retrieves the number of available buffers
double value = Ch.GetBuffersCount();
```

3.34.6 GetAxesCount

Description

The method returns the number of available axes.

Syntax

object.GetAxesCount ()

Async Syntax

object.GetAxesCountAsync ()

Arguments

None

Return Value

None

Receives the number of available axes.

Example

```
// Retrieves the number of available axes
double value = Ch.GetAxesCount();
```

3.34.7 GetDBufferIndex

Description

The method returns the index of the D-Buffer.

Syntax

object.GetDBufferIndex();

Async Syntax

ACSC_WAITBLOCK object.GetDBufferIndexAsync();

Arguments

None

Return Value

Double

Receives the D-Buffer Index

Example

```
// Retrieves the D-Buffer index
double value = Ch.GetDBufferIndex();
```

3.34.8 GetUMDVersion

Description

The method retrieves the Universal Mode Driver version as a string.

Syntax

object.UMDVersion()

Arguments

None

Return Value

Double

The method retrieves the Universal Mode Driver version.

Remarks

If the method fails, the Error object is filled with the Error Description.

Example

```
// Retrieves the firmware version of the controller
double UMDversion= api.GetUMDVersion();
```

3.35 Error Diagnostics Methods

The Error Diagnostics methods are:

Table 3-37. Error Diagnostics Methods

Method	Description
GetMotorError	Retrieves the reason why the motor was disabled.
GetMotionError	Retrieves the termination code of the last executed motion of the specified axis.
GetProgramError	Retrieves the error code of the last program error encountered in the specified buffer.

3.35.1 GetMotorError

Description

The method retrieves the reason for motor disabling.

Syntax

object.GetMotorError (Axisaxis)

Async Syntax

object.GetMotorErrorAsync (Axisaxis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

Int32.

The method retrieves the reason for the motor becoming disabled.

Remarks

If the motor is enabled the method returns zero. If the motor was disabled, the method returns the reason for the disabling. To get the error explanation, use the *GetErrorString* method.

See *SPiiPlus ACSPL+ Programmer's Guide* for all available motor error code descriptions. If the method fails, the Error object is filled with the Error Description.

Example

```
// Retrieves the reason for motor disabling
int mError = Ch.GetMotorError(Axis.ACSC_AXIS_0);
```

3.35.2 GetMotionError

Description

The method retrieves the termination code of the last executed motion of the specified axis.

Syntax

object.GetMotionError (Axis axis)

Async Syntax

object.GetMotionErrorAsync (Axis axis)

Arguments

axis	Axis constant: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

Int32.

The method retrieves the termination code of the last executed motion of the specified axis.

Remarks

If the motion is in progress, the method returns zero. If the motion terminates for any reason, the method returns the termination code. To get the error explanation, use the method *GetErrorString*.

See the *SPiiPlus ACSPL+ Programmer's Guide* for all available motion termination codes description. If the method fails, the Error object is filled with the Error Description.

Example

```
// Retrieves the termination code of the last executed motion of
// axis 0
int mError = Ch.GetMotionError(Axis.ACSC_AXIS_0);
```

3.35.3 GetProgramError

Description

The method retrieves the error code of the last program error encountered in thespecified buffer.

Syntax

object.GetProgramError (ProgramBuffer buffer)

Async Syntax

object.GetProgramErrorAsync (ProgramBuffer buffer)

Arguments

buffer	Number of the program buffer.

Return Value

Int32.

The method retrieves the error code of the last program error encountered in thespecified buffer.

Remarks

If the program is running, the method returns zero. If the program terminates for any reason, the method returns the termination code. To get the error explanation, use *GetErrorString*.

If the method fails, the Error object is filled with the Error Description.

Example

```
// Appends buffer 0 with 1 line
Ch.AppendBuffer(ProgramBuffer.ACSC_BUFFER_0,
   "!The program finished without STOP command");
// Run buffer 0
Ch.RunBuffer(ProgramBuffer.ACSC_BUFFER_0,null);
// Retrieves error 3114
int pError = Ch.GetProgramError(ProgramBuffer.ACSC_BUFFER_0);
```

3.36 Position Event Generation (PEG) Methods

Table 3-38. Position Event Generation (PEG) Methods

Method	Description
AssignPegNT	Assigns engine-to-encoder as well as additional digital outputs for use as PEG State and PEG Pulse outputs.
AssignPegOutputsNT	Sets output pins assignment and mapping between FGP_OUT signals to the bits of the ACSPL+ OUT(x) variable.
AssignFastInputsNT	Used for switching MARK_1 physical inputs to ACSPL+ variables as Fast General Purpose inputs.
PegIncNTV2	Sets the parameters for the Incremental PEG mode.
PegRandomNTV2	Sets the parameters for the Random PEG mode.
WaitPegReadyNT	Waits for the all values to be loaded and the PEG engine to be ready to respond to movement.
StartPegNT	Initiates the PEG process on the specified axis.
StopPegNT	Terminates the PEG process immediately on the specified axis.

3.36.1 AssignPegNT

Description

The method is used for engine-to-encoder assignment as well as for assigning additional digital outputs assignment for use as PEG State and PEG Pulse outputs specifically for the SPiiPlus family controllers.

Syntax

object.AssignPegNT(Axis axis, int engToEncBitCode, int gpOutsBitCode)

Async Syntax

ACSC_WAITBLOCK object.AssignPegNTAsync(Axis axis, int engToEncBitCode, int gpOutsBitCode)

Arguments

axis	PEG axis: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
engToEncBitCode	Bit code for engines-to-encoders mapping, see the *PEG and MARK Operations Application Notes*.
gpOutsBitCode	General Purpose outputs assignment to use as PEG state and PEG pulse, see the *PEG and MARK Operations Application Notes*.

Return Value

None

Example

```
// Example synchronous call to AssignPegNT
int engToEncBitCode =0x0;
int gpOutsBitCode = 0x0b11;
Ch.AssignPegNT(Axis.ACSC_AXIS_1, engToEncBitCode, gpOutsBitCode);
```

3.36.2 AssignPegOutputsNT

Description

The method is used for setting output pins assignment and mapping between **FGP_OUT** signals to the bits of the ACSPL+ **OUT(x)** variable, where x is the index that has been assigned to the controller in the network during System Configuration, specifically for the SPiiPlus family controllers.

OUT is an integer array that can be used for reading or writing the current state of the General Purpose outputs - see the *SPiiPlus Command & Variable Reference Guide*.

Syntax

object.AssignPegOutputsNT(Axis axis, int outputIndex, int bitCode)

Async Syntax

ACSC_WAITBLOCK object.AssignPegOutputsNTAsync(Axis axis, int outputIndex, int bitCode)

Arguments

axis	PEG axis: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
outputIndex	0 for **OUT_0**, 1 for **OUT_1**, ..., 9 for **OUT_9**
bitCode	Bit code for engine outputs to physical outputs mapping, see the *PEG and MARK Operations Application Note*.

Return Value

None

Example

```
// Example synchronous call to AssignPegOutputNT
int outputBitCode =0x0b0;
int outputIndex = 0;
Ch.AssignPegNT(Axis.ACSC_AXIS_1, outputIndex, outputBitCode);
```

3.36.3 AssignFastInputsNT

Description

The method is used to switch MARK_1 physical inputs to ACSPL+ variables as fast General Purpose inputs for SPiiPlus family controllers.

> While this method is not related to PEG activity, it is included with the PEG methods for the sake of completeness, since many times fast inputs are used in applications that use PEG functionality.

The method is used for setting input pins assignment and mapping between **FGP_IN** signals to the bits of the ACSPL+ **IN**(x) variable, where x is the index that has been assigned to the controller in the network during System Configuration.

IN is an integer array that can be used for reading the current state of the General Purpose inputs

- see the *SPiiPlus Command& Variable Reference Guide*.

Syntax

object.AssignFastInputsNT (Axis axis, int inputIndex, int bitCode)

Async Syntax

ACSC_WAITBLOCK object. AssignFastInputsNTAsync(Axis axis, int inputIndex, int bitCode)

Arguments

axis	PEG axis: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
inputIndex	0 for **IN_0**, 1 for **IN_1**, ..., 9 for **IN_9**
bitCode	Bit code for engine outputs to physical outputs mapping, see the *PEG and MARK Operations Application Note*.

Return Value

None

Example

```
// Example synchronous call to AssignPegOutputNT
Ch.AssignFastInputsNT(Axis.ACSC_AXIS_1, 1, 7);
```

3.36.4 PegIncNTV2

Description

The method is used for setting the parameters for the Incremental PEG mode for SPiiPlus family controllers. Incremental PEG is defined by first point, last point and the interval.

Syntax

public void PegIncNTV2(MotionFlags flags, PEG Engine, double width, double firstPoint, double interval, double lastPoint, int errMapAxis1, double axisCoord1, int errMapAxis2, double axisCoord2, int errMapMaxSize, double minDirDistance, int tbNumber, double tbPeriod)

Async Syntax

public ACSC_WAITBLOCK PegIncNTV2Async(MotionFlags flags, PEG Engine, double width, double firstPoint, double interval, double lastPoint, int errMapAxis1, double axisCoord1, int errMapAxis2, double axisCoord2, int errMapMaxSize, double minDirDistance, int tbNumber, double tbPeriod)

Arguments

flags	Bit-mapped parameter that can include following flags: > **ACSC_AMF_WAIT** > **ACSC_AMF_INVERT_OUTPUT** > **ACSC_AMF_ACCURATE** > **ACSC_AMF_SYNCHRONOUS** > **ACSC_AMF_ENDLESS** > **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION1D** > **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION2D** > **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION3D** > **ACSC_AMF_MAXIMUM_ARR_SIZE** > **ACSC_AMF_MIN_AXIS_DIRECTION** See flags table below for details.
axis	PEG axis: **ACSC_AXIS_0** corresponds to axis 0, **ACSC_AXIS_1** to axis 1, etc. For the axis constants see Axis Definitions.
width	Width of desired pulse in milliseconds.
firstPoint	Position where the first pulse is generated.
interval	Distance between the pulse-generating points.

lastPoint	Position where the last pulse is generated. If the **ACSC_AMF_ENDLESS** flag is declared, then this parameter should be set to **ACSC_NONE**.
errMapAxis1	The index of the first static axis when error mapping is used with PEG. Use with the **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION2D** flag alone or in combination with **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION3D** .
axisCoord1	The predefined location value of the first static axis when error map correction compensation is used with PEG. Use with the **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION2D** flag alone or in combination with **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION3D** .
errMapAxis2	The index of the second static axis when error mapping is used with PEG. Use with the **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION2D** and **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION3D** flags.
axisCoord2	The predefined location value of the second static axis when error map correction compensation is used with PEG. Use with the **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION2D** and **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION3D** flags.
errMapMaxSize	Maximum array size contained error correction data. Default is 512. Range from 1 to XARRSIZE. A negative value or 0 is interpreted as default.
minDirDistance	Limit defining actual motion as opposed to jitter. If motion along the axis is greater than this value, that motion is used to determine the direction of motion.
tbNumber	Number of time-based pulses generated after each encoder-based pulse.
tbPeriod	Period of time-based pulses.

Flags

Flag Name	Flag Meaning
ACSC_AMF_WAIT	The execution of the PEG is delayed until the **ACSC_STARTPEGNT** function is executed.
ACSC_AMF_INVERT_OUTPUT	The PEG pulse output is inverted.

Flag Name	Flag Meaning
ACSC_AMF_ACCURATE	The error accumulation is prevented by taking into account the rounding of the distance between incremental PEG events. You must use this flag if *interval* does not match a whole number of encoder counts. Using this switch is recommended for any application that uses the **acsc_PegIncNTV2** command, whether or not *interval* defines a whole number of encoder counts.
ACSC_AMF_SYNCHRONOUS	PEG starts synchronously with the motion sequence.
ACSC_AMF_ENDLESS	This flag supports endless incremental PEG. If the flag is set in in the acsc_PegIncNTV2 function, the last_point parameter is optional and ignored now , and the PEG never stops by position. To stop PEG use the **acsc_StopPegNT**command.
ACSC_AMF_DYNAMIC_ERROR_COMPENSATION1D	This flag supports 1D/2D dynamic error compensation for Incremental PEG. 1D/2D error compensation can be defined by the 1D error mapping functions documented in the Dynamic Error Compensation section.

ACSC_AMF_DYNAMIC_ERROR_COMPENSATION2D	This flag supports 2D dynamic error compensation for Incremental PEG.
	See Dynamic Error Compensation for dynamic error compensation function details.
	This flag requires 2 additional function arguments:
	`ErrMapAxis1`
	and
	`AxisCoord1`
	.

Flag Name	Flag Meaning

ACSC_AMF_DYNAMIC_ERROR_COMPENSATION3D	This flag supports 3D dynamic error compensation for Incremental PEG.
	See Dynamic Error Compensation for dynamic error compensation function details. This flag must be used in combination with **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION2D** and requires 4 additional function arguments:
	`ErrMapAxis1`
	,
	`AxisCoord1`
	,
	`ErrMapAxis2`
	, and
	`AxisCoord2`
	.

ACSC_AMF_MAXIMUM_ARR_SIZE	This flag supports 1D/2D dynamic error compensation for Incremental PEG. The `ErrMapMaxSize` parameter must have a relevant value with this flag is set.
ACSC_AMF_MIN_AXIS_DIRECTION	This flag signals that the `MinDirDistance` parameter indicates a value defining actual motion as opposed to jitter. If motion along the axis when error compensation is in force is greater than this value, that motion is used to determine the direction of motion.

Return Value

None

Synchronous Example

```
        Api acsApi = new Api();

       int DBufferIdx = ((int)acsApi.GetDBufferIndex());
        ProgramBuffer DBuffer = (ProgramBuffer)DBufferIdx;
        // Clearing all buffers execpt DBuffer.
        for (int i = 0; i < DBufferIdx - 1; i++)
        {
            acsApi.ClearBuffer((ProgramBuffer)i, 0, 5000);
        }

        // Clearing DBuffer
```

```
        acsApi.ClearBuffer(DBuffer, 5, 1000);
        acsApi.CompileBuffer(DBuffer);

        Axis axis0 = Axis.ACSC_AXIS_0; // Axis 0 that is going to be used for PEG.
        double Width = 0.01; // Pulse Width
        double FirstPoint = 1000; // Position where the first pulse is generated
        double Interval = 1000; // Distance between the pulse-generating points.
        double LastPoint = 10000; // Position where the last pulse is generated.

        acsApi.AssignPegNT(axis0, 0x00000100, 0b000); // Assign engine to encoder
        acsApi.AssignPegOutputsNT(axis0, 8, 0b0000); // Settings output pins assignment and
mapping between FGP_OUT signals to the bits of ACSPL+ OUT(x) variable.


          acsApi.PegIncNTV2(
                       MotionFlags.ACSC_AMF_DYNAMIC_ERROR_COMPENSATION2D |
                       MotionFlags.ACSC_AMF_DYNAMIC_ERROR_COMPENSATION3D,
                       axis0,
                       Width,
                       FirstPoint,
                       Interval,
                       LastPoint,
                       1,
                       10,
                       2,
                       20,
                       0,
                       -1,
                       -1,
                       -1);


            acsApi.WaitPegReadyNT(axis0, 10000); // Wait for the all values to be loaded and
                                                // the PEG engine to be ready to respond.

            // Enable Axis 0
            acsApi.Enable(axis0);
            acsApi.WaitMotorEnabled(axis0, 1, 5000);

            // Commut Axis 0
            acsApi.Commut(axis0);
            acsApi.WaitMotorCommutated(axis0, 1, 5000);

            // Setting the feedback position to 0
            acsApi.SetFPosition(axis0, 0);

            // Preforming PTP motion to 10000 point
            acsApi.ToPoint(MotionFlags.ACSC_NONE, axis0, 10000);

            // Wait for the motion to end
            acsApi.WaitMotionEnd(axis0, 5000);

            // Get the pulse counter of the PEG engine.
            string reply = acsApi.Transaction("?GETPEGCOUNT(0)").Trim();
```

Asynchronous Example

```
        Api acsApi = new Api();

       int DBufferIdx = ((int)acsApi.GetDBufferIndex());
```

```
        ProgramBuffer DBuffer = (ProgramBuffer)DBufferIdx;
        // Clearing all buffers execpt DBuffer.
        for (int i = 0; i < DBufferIdx - 1; i++)
        {
            acsApi.ClearBuffer((ProgramBuffer)i, 0, 5000);
        }

        // Clearing DBuffer
        acsApi.ClearBuffer(DBuffer, 5, 1000);
        acsApi.CompileBuffer(DBuffer);


        Axis axis0 = Axis.ACSC_AXIS_0; // Axis 0 that is going to be used for PEG.
        double Width = 0.01; // Pulse Width
        double FirstPoint = 1000; // Position where the first pulse is generated
        double Interval = 1000; // Distance between the pulse-generating points.
        double LastPoint = 10000; // Position where the last pulse is generated.

        ACSC_WAITBLOCK wait;
        object retObj;

        acsApi.AssignPegNT(axis0, 0x00000100, 0b000); // Assign engine to encoder
        acsApi.AssignPegOutputsNT(axis0, 8, 0b0000); // Settings output pins assignment and
                                                  //mapping between FGP_OUT signals
                                                 // to the bits of ACSPL+ OUT(x) variable.


               wait = acsApi.PegIncNTV2Async(
                       MotionFlags.ACSC_AMF_DYNAMIC_ERROR_COMPENSATION2D |
                       MotionFlags.ACSC_AMF_DYNAMIC_ERROR_COMPENSATION3D,
                       axis0,
                       Width,
                       FirstPoint,
                       Interval,
                       LastPoint,
                       1,
                       10,
                       2,
                       20,
                       0,
                       -1,
                       -1,
                       -1);
                       retObj = acsApi.GetResult(wait, 5000);
                       Thread.Sleep(1000);


            acsApi.WaitPegReadyNT(axis0, 10000); // Wait for the all values to be loaded and
the PEG engine to be ready to respond.

            // Enable Axis 0
            acsApi.Enable(axis0);
            acsApi.WaitMotorEnabled(axis0, 1, 5000);

            // Commut Axis 0
            acsApi.Commut(axis0);
            acsApi.WaitMotorCommutated(axis0, 1, 5000);

            // Setting the feedback position to 0
            acsApi.SetFPosition(axis0, 0);
```

```
            // Preforming PTP motion to 10000 point
            acsApi.ToPoint(MotionFlags.ACSC_NONE, axis0, 10000);

            // Wait for the motion to end
            acsApi.WaitMotionEnd(axis0, 5000);

            // Get the pulse counter of the PEG engine.
            string reply = acsApi.Transaction("?GETPEGCOUNT(0)").Trim();
```

Supported Versions

This function is supported from V3.13 onwards.

3.36.5 PegRandomNTV2

Description

The method is used for setting the parameters for the Random PEG mode for SPiiPlus family controllers. Random PEG function specifies an array of points where position-based events should be generated.

Syntax

public void PegRandomNTV2(MotionFlags flags, Axis axis, double width, int mode, int firstIndex, int lastIndex, string pointArray, string stateArray, int errMapAxis1, double axisCoord1, int errMapAxis2, double axisCoord2, double minDirDistance, int tbNumber, double tbPeriod)

Async Syntax

public ACSC_WAITBLOCK PegRandomNTV2Async(MotionFlags flags, Axis axis, double width, int mode, int firstIndex, int lastIndex, string pointArray, string stateArray, int errMapAxis1, double axisCoord1, int errMapAxis2, double axisCoord2, double minDirDistance, int tbNumber, double tbPeriod)

Arguments

flags	Bit-mapped parameter that can include following flags:
	ACSC_AMF_WAIT the execution of the PEG is delayed until the StartPegNT function is executed.
	ACSC_AMF_INVERT_OUTPUT the PEG pulse output is inverted.
	ACSC_AMF_SYNCHRONOUS PEG starts synchronously with the motion sequence.
	ACSC_AMF_DYNAMICLOADINGGPEG
	If the ACSC_AMF_DYNAMICLOADINGGPEG flag is included, dynamic loading of positions is implemented.
	ACSC_AMF_MODULE :
	A new ACSC_AMF_MODULE flag is introduced to support modulo axis. The Positions Arrays loaded once, provides pulses every Modulo cycle. The Position values should be inside the Modulo range. The PEG engine must be assigned to Modulo axis.
	ACSC_AMF_DYNAMIC_ERROR_COMPENSATION1D
	This flag supports 1D/2D dynamic error compensation for Incremental PEG.
	1D/2D error compensation can be defined by the 1D error mapping functions documented in the Dynamic Error Compensation section.
	ACSC_AMF_DYNAMIC_ERROR_COMPENSATION2D
	This flag supports 2D dynamic error compensation for Incremental PEG.
	See Dynamic Error Compensation for dynamic error compensation function details.
	This flag requires 2 additional function arguments:
	`ErrMapAxis1`
	and
	`AxisCoord1`
	.
	ACSC_AMF_DYNAMIC_ERROR_COMPENSATION3D
	This flag supports 3D dynamic error compensation for Incremental PEG.
	See Dynamic Error Compensation for dynamic error compensation function details. This flag must be used in combination with **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION2D** and requires 4 additional function arguments:
	`ErrMapAxis1`
	,
	`AxisCoord1`
	,

	`ErrMapAxis2`
	, and
	`AxisCoord2`
	.
	ACSC_AMF_MIN_AXIS_DIRECTION
	This flag signals that the
	`MinDirDistance`
	parameter indicates a value defining actual motion as opposed to jitter. If motion along the axis when error compensation is in force is greater than this value, that motion is used to determine the direction of motion.

axis	PEG axis: **ACSC_AXIS_0** corresponds to axis 0, **ACSC_AXIS_1** to axis 1, etc. For the axis constants see Axis Definitions.
width	Width of desired pulse in milliseconds.
mode	Output signal configuration according to the ASSIGNPEG chapter in the *PEG and MARK Operations Application Notes.*.
firstIndex	Index of position in **PointArray** where the first pulse is generated.
lastIndex	Index of position in **PointArray** where the last pulse is generated.
pointArray	String containing the name of the real array that stores positions at which PEG pulse are to be generated The array must be declared as a global variable by an ACSPL+ program or by the DeclareVariable function.
stateArray	String containing the name of the integer array that stores desired output state at each position. The array must be declared as a global variable by an ACSPL+ program or by the **DeclareVariable** function. If output state change is not desired, this parameter should be NULL.
errMapAxis1	The index of the first static axis when error mapping is used with PEG. Use with the **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION2D** flag alone or in combination with **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION3D** .
axisCoord1	The predefined location value of the first static axis when error map correction compensation is used with PEG. Use with the **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION2D** flag alone or in combination with **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION3D** .

errMapAxis2	The index of the second static axis when error mapping is used with PEG. Use with the **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION2D** and **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION3D** flags.
axisCoord2	The predefined location value of the second static axis when error map correction compensation is used with PEG. Use with the **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION2D** and **ACSC_AMF_DYNAMIC_ERROR_COMPENSATION3D** flags.
minDirDistance	Limit defining actual motion as opposed to jitter. If motion along the axis is greater than this value, that motion is used to determine the direction of motion.
tbNumber	Number of time-based pulses generated after each encoder-based pulse.
tbPeriod	Period of time-based pulses.

Return Value

None

Synchronous Example

```
            Api acsApi = new Api();

                int DBufferIdx = (int)acsApi.GetDBufferIndex(); // Get DBuffer index
                ProgramBuffer DBuffer = (ProgramBuffer)DBufferIdx;

                // Clear all Buffers except DBuffer.
                for (int i = 0; i < DBufferIdx - 1; i++)
                {
                    acsApi.ClearBuffer((ProgramBuffer)i, 0, 5000);
                }

                // Clearing DBuffer
                acsApi.ClearBuffer(DBuffer, 5, 1000);
                acsApi.CompileBuffer(DBuffer);
                bool flag = false;
                int firstIndex = 0; // first index of point array.
                int lastIndex = 4; // last index of point array.
                double width = 0.02; // pulse width
                int mode = 0x4444; // mode - Output signal configuration according to
ASSIGNPEG.
                string pointArray = "PEG0_PositionArray1"; // Name of points array
                string stateArray = "PEG0_StatesArray1"; // Name of state array.
                acsApi.AppendBuffer(DBuffer, "GLOBAL REAL PEG0_PositionArray1(5)"); //
Appending the points array in DBuffer
                acsApi.AppendBuffer(DBuffer, "GLOBAL INT PEG0_StatesArray1(5)"); //
Appending the states array in DBuffer
                acsApi.CompileBuffer(DBuffer);

                // Filling the points array with values.
                for (int i = 0; i < 5; i++)
                {
                    int value = 100;
                    int temp = value * (i + 1);
```

```
                    value = temp;
                    acsApi.WriteVariable(value, "PEG0_PositionArray1", DBuffer, i, i, -1, -
1);
                }


                Axis axis0 = Axis.ACSC_AXIS_0; // Axis 0 that is going to be used for PEG.



                acsApi.AssignPegNT(axis0, 0x00000000, 0b000); // Assign engine to encoder
                // Setting output pins assignment and mapping between FGP_OUT signals to the
bits of ACSPL+ OUT(x) variable.
                acsApi.AssignPegOutputsNT(axis0, 8, 0b0000);
                acsApi.AssignPegOutputsNT(axis0, 0, 0b0000);
                acsApi.AssignPegOutputsNT(axis0, 1, 0b0000);
                acsApi.AssignPegOutputsNT(axis0, 2, 0b0000);
                acsApi.AssignPegOutputsNT(axis0, 3, 0b0000);

                acsApi.PegRandomNTV2Async(MotionFlags.ACSC_AMF_DYNAMIC_ERROR_COMPENSATION2D
| MotionFlags.ACSC_AMF_DYNAMIC_ERROR_COMPENSATION3D,
                    axis0,
                    width,
                    mode,
                    firstIndex,
                    lastIndex,
                    pointArray,
                    stateArray,
                    1,
                    10,
                    2,
                    20,
                    -1,
                    -1,
                    -1);


                acsApi.WaitPegReadyNT(axis0, 10000); // Wait for the all values to be loaded
and the PEG engine to be ready to respond.

                // Enable Axis 0
                acsApi.Enable(axis0);
                acsApi.WaitMotorEnabled(axis0, 1, 5000);

                // Commut Axis 0
                acsApi.Commut(axis0);
                acsApi.WaitMotorCommutated(axis0, 1, 5000);

                // Setting the feedback position to 0
                acsApi.SetFPosition(axis0, 0);

                // Preforming PTP motion to 10000 point.
                acsApi.ToPoint(MotionFlags.ACSC_NONE, axis0, 10000);

                // Wait for the motion to end.
                acsApi.WaitMotionEnd(axis0, 5000);

                // Get the pulse counter of the PEG engine.
                string PEGCount = acsApi.Transaction("?GETPEGCOUNT(0)").Trim();
```

Asynchronous Example

```
            Api acsApi = new Api();

                int DBufferIdx = (int)acsApi.GetDBufferIndex(); // Get DBuffer index
                ProgramBuffer DBuffer = (ProgramBuffer)DBufferIdx;

                // Clear all Buffers except DBuffer.
                for (int i = 0; i < DBufferIdx - 1; i++)
                {
                    acsApi.ClearBuffer((ProgramBuffer)i, 0, 5000);
                }

                // Clearing DBuffer
                acsApi.ClearBuffer(DBuffer, 5, 1000);
                acsApi.CompileBuffer(DBuffer);
                bool flag = false;
                int firstIndex = 0; // first index of point array.
                int lastIndex = 4; // last index of point array.
                double width = 0.02; // pulse width
                int mode = 0x4444; // mode - Output signal configuration according to
ASSIGNPEG.
                string pointArray = "PEG0_PositionArray1"; // Name of points array
                string stateArray = "PEG0_StatesArray1"; // Name of state array.
               // Appending the points array in DBuffer
                acsApi.AppendBuffer(DBuffer, "GLOBAL REAL PEG0_PositionArray1(5)");
               // Appending the states array in DBuffer
                acsApi.AppendBuffer(DBuffer, "GLOBAL INT PEG0_StatesArray1(5)");
                acsApi.CompileBuffer(DBuffer);

                // Filling the points array with values.
                for (int i = 0; i < 5; i++)
                {
                    int value = 100;
                    int temp = value * (i + 1);
                    value = temp;
                    acsApi.WriteVariable(value, "PEG0_PositionArray1", DBuffer, i, i, -1, -
1);
                }


                Axis axis0 = Axis.ACSC_AXIS_0; // Axis 0 that is going to be used for PEG.

               ACSC_WAITBLOCK wait;
               object retObj;

                acsApi.AssignPegNT(axis0, 0x00000000, 0b000); // Assign engine to encoder
                // Setting output pins assignment and mapping between FGP_OUT signals to the
bits of ACSPL+ OUT(x) variable.
                acsApi.AssignPegOutputsNT(axis0, 8, 0b0000);
                acsApi.AssignPegOutputsNT(axis0, 0, 0b0000);
                acsApi.AssignPegOutputsNT(axis0, 1, 0b0000);
                acsApi.AssignPegOutputsNT(axis0, 2, 0b0000);
                acsApi.AssignPegOutputsNT(axis0, 3, 0b0000);

                wait = acsApi.PegRandomNTV2Async(MotionFlags.ACSC_AMF_DYNAMIC_ERROR_
COMPENSATION2D | MotionFlags.ACSC_AMF_DYNAMIC_ERROR_COMPENSATION3D,
                    axis0,
                    width,
                    mode,
                    firstIndex,
```

```
                     lastIndex,
                     pointArray,
                     stateArray,
                     1,
                     10,
                     2,
                     20,
                     -1,
                     -1,
                     -1);
                 retObj = acsApi.GetResult(wait, 5000);
                 Thread.Sleep(500);

                 acsApi.WaitPegReadyNT(axis0, 10000); // Wait for the all values to be loaded
and the PEG engine to be ready to respond.

                 // Enable Axis 0
                 acsApi.Enable(axis0);
                 acsApi.WaitMotorEnabled(axis0, 1, 5000);

                 // Commut Axis 0
                 acsApi.Commut(axis0);
                 acsApi.WaitMotorCommutated(axis0, 1, 5000);

                 // Setting the feedback position to 0
                 acsApi.SetFPosition(axis0, 0);

                 // Preforming PTP motion to 10000 point.
                 acsApi.ToPoint(MotionFlags.ACSC_NONE, axis0, 10000);

                 // Wait for the motion to end.
                 acsApi.WaitMotionEnd(axis0, 5000);

                 // Get the pulse counter of the PEG engine.
                 string PEGCount = acsApi.Transaction("?GETPEGCOUNT(0)").Trim();
```

Supported Versions

This function is supported from V3.13 onwards.

3.36.6 WaitPegReadyNT

Description

The method waits for the all values to be loaded and the PEG engine to be ready to respond to movement on the specified axis for SPiiPlus familycontrollers.

The method can be used in both the Incremental and Random PEG modes.

Syntax

object.WaitPegReadyNT (Axis axis, int timeout)

Async Syntax

object.WaitPegReadyNTAsync (Axis axis, int timeout)

Arguments

axis	PEG axis: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.
timeout	Maximum waiting time in milliseconds. If **timeout** is INFINITE, the method's time-out interval never elapses.

Return Value

None

Example

```
// Example synchronous call to WaitPegReadyNT
int timeout = 2000;
Ch.WaitPegReadyNT(Axis.ACSC_AXIS_1,timeout);
```

3.36.7 StartPegNT

Description

The method is used to initiate the PEG process on the specified axis for SPiiPlus family controllers.

Syntax

object.StartPegNT(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.StartPegNTAsync(Axis axis)

Arguments

axis	PEG axis: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

None

Example

```
// Example synchronous call to StartPegNT
Ch.StartPegNT(Axis.ACSC_AXIS_1);
```

3.36.8 StopPegNT

Description

The method is used to terminate the PEG process immediately on the specified axis for SPiiPlus family controllers.

The method can be used in both the Incremental and Random PEG modes.

Syntax

object.StopPegNT(Axis axis)

Async Syntax

ACSC_WAITBLOCK object.StopPegNTAsync(Axis axis)

Arguments

Axis	PEG axis: ACSC_AXIS_0 corresponds to the axis 0, ACSC_AXIS_1 to the axis 1, etc. For the axis constants see Axis Definitions.

Return Value

None

Example

```
// Example synchronous call to StopPegNT
Ch.StopPegNT(Axis.ACSC_AXIS_1);
```

3.37 Application Save/Load Methods

The Application Save/Load methods are:

Table 3-39. Application Save/Load Methods

Method	Description
AnalyzeApplication	Analyzes the type of application
LoadApplication	Loads the applicationi
SaveApplication	Saves the application
FreeApplication	Frees memory after application is taken off

3.37.1 Structures and Classes

The Application Loader methods employ the following structures and classes

3.37.1.1 ApplicationFileInfo

Description

This structure describes any sections (or controller files).

Syntax

```
struct ACSC_APPSL_SECTION
{
   ACSC_APPSL_FILETYPE type;
   ACSC_APPSL_STRING filename;
   ACSC_APPSL_STRING Description;
   UInt32 size;
   UInt32 offset;
```

```
            UInt32 CRC;
            Int32 inuse;
            Int32 error;
            IntPtr pData;
        }
```

Variables

type	Section (controller file) type
filename	Section (controller file) name
Description	Section (controller file) *Description*
size	Data size
offset	Offset in the file data section
CRC	Data CRC
inuse	1 - In use 0 - Not in use
error	Error code
data	Pointer to start of data

Properties

Data	Data string

3.37.1.2 ACSC_APPSL_STRING

Description

This structure used for Application Saver / Loader functions.

Syntax

struct ACSC_APPSL_STRING

Properties

Value	String value

3.37.2 Enumerations

The Application Loader methods employ the following enums:

3.37.2.1 ACSC_APPSL_FILETYPE

Description

Describes possible file types.

Syntax

```
public enum ACSC_APPSL_FILETYPE
{
   ACSC_ADJ,
   ACSC_SP,
   ACSC_ACSPL,
   ACSC_PAR,
   ACSC_USER
}
```

Values

ACSC_ADJ	Value 0: File type is an Adjuster
ACSC_SP	Value 1: File type is an SP application
ACSC_ACSPL	Value 2: File type is a Program Buffer
ACSC_PAR	Value 3: File type is a Parameters file.
ACSC_USER	Value 4: File type is a User file

3.37.3 AnalyzeApplication

Description

The method analyzes application file and returns information about the file components, such as, saved ACSPL+ programs, configuration parameters, user files, etc.

Syntax

object.AnalyzeApplication (string fileName)

Arguments

fileName	Variable that specifies path to host application file. If analyzing controller application, this parameter should be NULL or empty string.

Return Value

Return Value is **class ApplicationFileInfo** described in *ApplicationFileInfo*. If the method fails, there is exception thrown.

Example

```
ApplicationFileInfo info;
string fileName = "neededFile";
info = Ch.AnalyzeApplication(fileName);
string strOut = Ch.LoadApplication(fileName, info, false);
```

3.37.4 LoadApplication

Description

The method loads user application file from a host PC to the controller flash.

Syntax

object.LoadApplication (string fileName, ApplicationFileInfo info, bool isPreview)

Arguments

fileName	Variable that specifies path to host application file.
info	Class that describes all the application file data.
isPreview	For internal use only. It must always be 0.

Return Value

String – reserved for internal use

Comments

The user should reboot the controller after the function operation to update the controller configuration.

To reboot the controller the user can use the ControllerReboot function.

WARNING

The flash memory is rated to support about 100,000 write operations.
ACS recommends against frequent, repeated writing to the flash memory.

Example

```
ApplicationFileInfo info;
string fileName = "neededFile";
info = Ch.AnalyzeApplication(fileName);
string strOut = Ch.LoadApplication(fileName, info, false);
Ch.ControllerReboot(30000);
```

3.37.5 SaveApplication

Description

The method saves user application from the controller to a file on host PC.

Syntax:

object.SaveApplication (string fileName, ApplicationFileInfo info, bool isPreview)

Arguments

fileName	Variable that specifies path to host application file.
info	Class that describes all the application file data.

Return Value

String – reserved for internal use

Example

```
string filename = "neededFile";
ApplicationFileInfo info = Ch.AnalyzeApplication(null);
Ch.SaveApplication(filename, info);
```

3.37.6 FreeApplication

Description

The method frees memory, previously allocated by the AnalyzeApplication method.

Syntax

object.FreeApplication (ApplicationFileInfo info)

Arguments

info	Class that describes all the application file data.

Return Value

None

Example

```
string filename = "neededFile";
ApplicationFileInfo info = Ch.AnalyzeApplication(filename);
Ch.FreeApplication(info);
```

3.38 Load/Upload Data To/From Controller Methods

The Load/Upload Data To/From Controller methods are:

Table 3-40. Load/Upload Data To/From Controller Methods

Method	Description
LoadDataToController	Writes value(s) from text file to SPiiPlus controller (variable or file).
UploadDataFromController	Writes value(s) from SPiiPlus controller (variable or file) to text file.

3.38.1 LoadDataToController

Description

This method writes value(s) from text file to SPiiPlus controller (variable or file).

Syntax:

object.LoadDataToController(int dest, string destName, int from1, int to1,int from2, int to2, string srcFilename, int srcNumFormat, bool bTranspose)

Async Syntax:

ACSC_WAITBLOCK object.LoadDataToControllerAsync(int dest, string destName, int from1, int to1,int from2, int to2, string srcFilename, int srcNumFormat, bool bTranspose)

Arguments

dest	Number of program buffer for local variable **Api.ACSC_NONE** for global and ACSPL+ variable **GeneralDefinition.ACSC_FILE** for loading directly to file on flash memory (only arrays can be written directly into controller files)
destName	String that contains name of the Variable or File
from1/to1	Index range (first dimension)
from2/to2	Index range (second dimension)
srcFilename	Filename (including path) of the source text data file
srcNumFormat	Format of number(s) in source file. Use: **GeneralDefinition.ACSC_INT_BINARY** for integers, and **GeneralDefinition.ACSC_REAL_BINARY** for real
bTranspose	If TRUE (1), then the array will be transposed before being loaded. Otherwise, this parameter has no affect

Return Value

None

Remarks

The method writes to a specified variable (scalar/array) or straight to binary file on controller's flash memory. The variable can be a ACSPL+ controller variable, user global or user local. The input file must be ANSI format, otherwise an error 168 (invalid file format) is returned.

ACSPL+ and user global variables have global scope. Therefore, **dest** must be Api.ACSC_NONE (1) for these classes of variables. User local variable exists only within a buffer. The buffer number must be specified for user local variable.

If **dest** is Api.ACSC_NONE (-1) and there is no global variable with the name specified by **destName**, it will be defined. Arrays will be defined with dimensions (**to1**+1,**to2**+1). If performing loading straight to file, **from1**, **to1**, **from2,** and **to2** are meaningless.

If the variable is scalar, all indexes **from1**, **to1**, **from2**, and **to2** must be Api.ACSC_NONE (-1). The method writes the value from file specified by **SrcFileName**, to the variable specified by **name**. In this case if the variable is a one-dimensional array, **from1**, **to1** must specify the index range and **from2**, **to2** must be **Api.ACSC_NONE** (-1). The text file, pointed to by **srcFileName**, must contain **to1-from1** +1 values at least. The method writes the values to the specified variable from index **from1** to index **to1** inclusively.

If the variable is a two-dimensional array, **from1**, **to1** must specify the index range of the first dimension and **from2**, **to2** must specify the index range of the second dimension. The text file, pointed to by **srcFileName**, must contain ((**to1-from1** +1) x (**to2-from2** +1)) values at least. Otherwise, error will occur.

The method uses the text file as follows: first, the method retrieves **the to2-from2** +1 values and writes them to row **from1** of the specified controller variable, then retrieves next to**2- from2** +1 values and writes them to row **from1** +1 of the specified controller variable, and so forth. If **bTranspose** is TRUE, the method actions are inverted. It takes **to1-from1** +1 values and writes them to column **from2** of the specified controller variable, then retrieves next **to1- from1** +1 values and writes them to column **from2** +1 of the specified controller variable, and so forth.

The text file is processed line-by-line; any characters except numbers, dots, commas and exponent 'e' are translated as separators between the numbers. Line that starts with no digits is considered as comment and ignored.

WARNING	The flash memory is rated to support about 100,000 write operations. ACS recommends against frequent, repeated writing to the flash memory.

Example

```
// Example call LoadDataToController
Ch.LoadDataToController((int)GeneralDefinition.ACSC_FILE,
"MyArrayInFile", Api.ACSC_NONE, Api.ACSC_NONE, Api.ACSC_NONE,
Api.ACSC_NONE,"C:\\UserArray.txt",(int)GeneralDefinition.ACSC_INT_BINARY,
false);
```

3.38.2 UploadDataFromController

Description

This method writes value(s) from SPiiPlus controller (variable or file) to text file.

Syntax:

object.UploadDataFromController (int src, string srcName, int srcNumFormat, int from1, int to1, int from2, int to2, string destFilename, string destNumFormat, Boolean bTranspose)

Async Syntax

object.UploadDataFromControllerAsync (int src, string srcName, int srcNumFormat, int from1, int to1, int from2, int to2, string destFilename, string destNumFormat, Boolean bTranspose)

Arguments

src	Number of program buffer for local variable, **Api.ACSC_NONE** for global and ACSPL+ variable and **GeneralDefinition.ACSC_FILE** for loading directly from file on flash memory.
srcName	String that contains name of the Variable or File
srcNumFormat	Format of number(s) in Controller. Use **GeneralDefinition.ACSC_INT_BINARY** for integer, and **GeneralDefinition.ACSC_REAL_BINARY** for real
from1/to1	Index range (first dimension)
from2/to2	Index range (second dimension)
destFilename	Filename (including path) of the source text data file
destNumFormat	Formatting string that will be used for printing into file ("1:%d\n" for *Example*). Use string with %d for integer, and %lf for real type
bTranspose	If TRUE (1), then the array will be transposed before being loaded. Otherwise, this parameter has no affect

Return Value

None

Remarks

The method writes data to file from a specified variable (scalar/array) or straight from a binary file in the controller's flash memory. The variable can be a ACSPL+ controller variable, user global or user local.

ACSPL+ and user global variables have global scope. Therefore **src** must have the value of Api.ACSC_NONE (-1) for these classes of variables. User local variable exists only within a buffer. The buffer number must be specified for user localvariable.

If there is no variable (or file) with the name specified by **srcName**, there would be error. If performing loading straight to file, **from1**, **to1**, **from2,** and **to2** are meaningless.

If the variable is scalar, all indexes **from1**, **to1**, **from2**, and **to2** must be Api.ACSC_NONE (-1). The method writes the value from variable specified by **srcName**, to the file specified by **destFileName**.

If the variable is a one-dimensional array, **from1**, **to1** must specify the index range and **from2**, **to2** must be Api.ACSC_NONE (-1). The method writes the values from the specified variable from index from1 to index **to1** inclusively, to the file specified **bydestFileName**.

If the variable is a two-dimensional array, **from1**, **to1** must specify the index range of the first dimension and **from2**, **to2** must specify the index range of the second dimension.

The method uses the variable as follows: first, the method retrieves the **to2-from2** +1 values from row **from1** and writes them to the file specified by **destFileName**, then retrievesto**2- from2** +1 values from row **from1** +1 and writes them, and so forth.

If **bTranspose** is TRUE, the method actions are inverted. It takes **to1-from1** +1 values from row **from2** and writes them to first column of the specified controller variable, then retrieves the next **to1-from1** values from row **from2** +1 and writes them to the ext column of the specified destination file.

The destination file's format will be determined by string specified by **destNumFormat**. This string will be used as argument in the ***printf** function.

Example

```
// Example call UploadDataFromController
Ch.UploadDataFromController(Api.ACSC_NONE, "MyGlobalArray",
(int)GeneralDefinition.ACSC_INT_BINARY, 0, 3, 2,
4,"C:\\MyTransposedArrayInFile.txt", "%d", true);
```

3.39 Emergency Stop Methods

The Emergency Stop methods are:

Table 3-41. Emergency Stop Methods

Method	Description
RegisterEmergencyStop	Enables Emergency Stop function.
UnregisterEmergencyStop	Disables Emergency Stop function.

3.39.1 RegisterEmergencyStop

Description

This method initiates the "Emergency Stop" functionality for the calling application.

Syntax

object.RegisterEmergencyStop();

Return Value

None

Remarks

SPiiPlusUMD and Library provide the user application with the ability to open/close the Emergency Stop button (shown below). Clicking the Emergency Stop button sends a **stop** to all motions and motors command to all channels, which used in calling application, thereby stopping all motions and disabling all motors.

Figure 3-1. Emergency Stop Button

Calling **RegisterEmergencyStop** causes the Emergency Stop button icon to appear in the right bottom corner of the screen. If there is already such a button that is in use by other applications, a new button does not appear; but all functionality is available for the new application.

Calling **RegisterEmergencyStop** requires having the local host SPiiPlus UMD running, even if it is used through a remote connection because the Emergency Stop button is part of the local SPiiPlus UMD. If there is no local SPiiPlus UMD running, the method fails.

Only a single call is required per application. It can be placed anywhere in code, even before the opening of communication with controllers.

The application can remove the Emergency Stop button by calling *UnregisterEmergencyStop*. The Emergency Stop button disappears if there are no additional registered applications that use it. The termination of SPiiPlus UMD also removes the Emergency Stop button; therefore, if SPiiPlus UMD is restarted, **RegisterEmergencyStop** has to be called again.

Calling **RegisterEmergencyStop** more than once per application is meaningless, but the method succeeds anyway. In order to ensure that Emergency Stop button is active, itis recommended placing a call of **RegisterEmergencyStop** after each call of any of **OpenComm***** functions.

Example

```
// Example call Register Emergency stop
Ch.RegisterEmergencyStop();
```

3.39.2 UnregisterEmergencyStop

Description

This method terminates the “Emergency Stop” functionality for the calling application.

Syntax

object.UnregisterEmergencyStop()

Return Value

None

Remarks

Calling **UnregisterEmergencyStop** causes an application not to respond if the Emergency Stop button is clicked. If there are no other applications that registered the EmergencyStop functionality, the button will disappear.

Calling **UnregisterEmergencyStop** more than once per application is meaningless, but method will succeed anyway.

Example

```
// Example call Unregister Emergency stop
Ch.UnregisterEmergencyStop();
```

3.40 Reboot Methods

The Reboot methods are:

Table 3-42. Reboot Methods

Method	Description
ControllerReboot	Reboots controller and waits for process completion.
ControllerFactoryDefault	Reboots controller, restores factory default settings and waits for process completion.

3.40.1 ControllerReboot

Description

This method reboots controller and waits for processcompletion.

Syntax

object.ControllerReboot (int timeout)

Arguments

timeout	Maximum waiting time in milliseconds.

Return Value

None

Example

```
// Open Ethernet with TCP protocol communication with the controller
// IP address: 10.0.0.100
EthernetCommOption port = EthernetCommOption.ACSC_SOCKET_STREAM_PORT;
Ch.OpenCommEthernetTCP("10.0.0.100", (int)port);
Ch.ControllerReboot(30000);
Ch.CloseComm();
```

3.40.2 ControllerFactoryDefault

Description

The method reboots the controller, restores factory default settings and waits for process completion.

Syntax

object.ControllerFactoryDefault (int timeout)

Arguments

timeout	Maximum waiting time in milliseconds.

Return Value

If the method succeeds, the return value is non-zero. If the method fails, the return value is zero.

Example

```
// Open Ethernet with TCP protocol communication with the controller
// IP address: 10.0.0.100
EthernetCommOption port = EthernetCommOption.ACSC_SOCKET_STREAM_PORT;
Ch.OpenCommEthernetTCP("10.0.0.100", (int)port);
Ch.ControllerFactoryDefault(30000);
Ch.CloseComm();
```

3.41 Host-Controller File Operations

Host PC files can be copied to controller's non-volatile memory and user files can be deleted from the controller's non-volatile memory as described in this section.

Table 3-43. Host-Controller File Copying Methods

Method	Description
CopyFileToController	The function copies files from the host PC to the controller's non-volatile memory
DeleteFileFromController	The function deletes user files from the controller's non-volatile memory.

3.41.1 CopyFileToController

Description

The function copies file from host PC to controller’s non-volatile memory.

Syntax

object.CopyFileToController (string sourceFileName, string destinationFileName)

Async Syntax

object.CopyFileToControllerAsync (string sourceFileName, string destinationFileName)

Arguments

sourceFileName	Pointer to a buffer that contains the name of the source file on host PC
destinationFileName	Pointer to a buffer that contains name of destination file on the controller.

Return Value

None

Example

```
// Example call CopyFileToController
 Ch.CopyFileToController("C:\\tmp.txt","tmp.txt");
```

3.41.2 DeleteFileFromController

Description

The function deletes user files from controller's non-volatile memory.

Syntax

object.DeleteFileFromController (string fileName)

Async Syntax

object.DeleteFileFromControllerAsync (string fileName)

Arguments

fileName	Pointer to a buffer that contains name of the user file on controller's non-volatile memory.

Return Value

None

Example

```
// Example call DeleteFileFromController
Ch.DeleteFileFromController("tmp.txt");
```

3.42 Save to Flash Functions

The Save to flash method is

Table 3-44. Save to Flash Methods

Method	Description
ControllerSaveToFlash	The function saves user application to the controller's non-volatile memory.

3.42.1 ControllerSaveToFlash

Description

The function saves user application to the controller's non-volatile memory.

Syntax

object.ControllerSaveToFlash (Axis[] parameters, ProgramBuffer[] buffers,ServoProcessor[] sPPrograms, string userArrays)

Arguments

parameters	Array of parameters constants. Each element specifies system parameters or one involved axis: **ACSC_SYSTEM** corresponds to system parameters; ACSC_AXIS_0 corresponds to axis 0, **ACSC_AXIS**_1 to axis 1, etc. If there is no need to save user arrays to controller's non-volatile memory, this parameter should be NULL.
buffers	Array of buffer constants. Each element specifies one involved buffer: **ACSC_BUFFER_0** corresponds to buffer 0, **ACSC_BUFFER_1** to buffer 1, etc. If all buffers need to be specified, **ACSC_BUFFER_ALL** should be used. After the last buffer, one additional element must be located that contains -1 which marks the end of the array.
sPPrograms	Array of Servo Processor (SP) constants. Each element specifies one involved SP: **ACSC_SP_0** corresponds to SP 0, **ACSC_SP_1** to SP 1, etc. If all SPs need to be specified, **ACSC_SP_ALL** should be used. After the last SP, one additional element must be located that contains - 1 which marks the end of the array.
userArrays	User Arrays list - Pointer to the string. The string contains chained names of user arrays, separated by '\r'(13) character. If there is no need to save user arrays to controller's non-volatile memory, this parameter should be **NULL**.

Return Value

Int32.

Example

The following code sample saves all axes parameters and buffers to flash, with two user arrays: "MyArray" and"MyArray2".

```
Axis[] parameters = new Axis[2];
ProgramBuffer[] buffers = new ProgramBuffer[2];
ServoProcessor[] sPPrograms = new ServoProcessor[2];
parameters[0] = Axis.ACSC_PAR_ALL;
parameters[1] = Axis.ACSC_NONE;
buffers[0] = ProgramBuffer.ACSC_BUFFER_ALL;
buffers[1] = ProgramBuffer.ACSC_NONE;
sPPrograms[0] = ServoProcessor.ACSC_SP_ALL;
sPPrograms[1] = ServoProcessor.ACSC_NONE;
Ch.ControllerSaveToFlash(parameters,
    buffers, sPPrograms,"MyArray\rMyArray2");
```

3.43 SPiiPlusSC Management

The SPiiPlusSC Management methods are:

Table 3-45. SPiiPlusSC Management Methods

Method	Description
StartSPiiPlusSC	The function starts the SPiiPlusSC controller.
StopSPiiPlusSC	The function stops the SPiiPlusSC controller.

3.43.1 StartSPiiPlusSC

Description

The function starts the SPiiPlusSC controller.

Syntax

object.StartSPiiPlusSC()

Arguments

None

Return Value

None

Example

```
// Example call StartSPiiPlusSC
Ch.StartSPiiPlusSC();
```

3.43.2 StopSPiiPlusSC

Description

The function stops the SPiiPlusSC controller.

Syntax

object.StopSPiiPlusSC()

Arguments

None

Return Value

None

Example

```
// Example call StopSPiiPlusSC
Ch.StopSPiiPlusSC();
```

3.44 FRF Methods

3.44.1 FRFMeasure

Description

This function initializes FRF measurement

Syntax

public object.FRFOutput FRFMeasure(FRFInput input);

Arguments

FRFInput	Input for FRF initialization

Return Value

FRFOutput object

Example

```
Api Ch = new Api();
Ch.OpenCommEthernet("10.0.0.100", 701);
FRFInput input = new FRFInput();
input.Axis = 0;
input.LoopType = FRF_LOOP_TYPE.PositionVelocity;
input.ExcitationType = FRF_EXCITATION_TYPE.WhiteNoise;
input.ChirpType = FRF_CHIRP_TYPE.LinearChirp;
input.WindowType = FRF_WINDOW_TYPE.Hanning;
input.FrequencyDistributionType = FRF_FREQUENCY_DISTRIBUTION_
TYPE.Logarithmic;
input.Overlap = FRF_OVERLAP.HalfSignal;
input.StartFreqHz = 3;
input.EndFreqHz = 3000;
input.FreqPerDec = 50;
input.HighResolutionStart = 1000;
input.HighResolutionFreqPerDec = 1000;
input.ExcitationAmplitudePercentIp = 5;
input.DurationSec = 1;
input.NumberOfRepetitions = 5;
input.Recalculate = false;
FRFOutput output = Ch.FRFMeasure(input);
```

3.44.2 FRFStop

Description

This function aborts FRF measurement.

Syntax

public void object.FRFStop(Axis axis);

Arguments

Axis	Axis to abort

Return Value

None

Example

```
Ch.FRFStop((Axis)0);
```

3.44.3 FRFCalculateDuration

Description

This function calculates the required duration of measurement (DurationSec parameter of FRFInput class) in order to satisfy specified frequency resolution.

Syntax

public double object.FRFCalculateDuration(FRF_DURATION_CALCULATION_PARAMETERS durationCalculationParameters)

Arguments

FRF_DURATION_CALCULATION_PARAMETERS	Object with duration calculation parameters

Return Value

Duration calculated

Example

```
FRF_DURATION_CALCULATION_PARAMETERS input = new FRF_DURATION_CALCULATION_
PARAMETERS();
input.StartFreqHz = 10;
input.EndFreqHz = 3000;
input.FreqPerDec = 50;
input.HighResolutionStart = 500;
input.HighResolutionFreqPerDec = 500;
input.FrequencyDistributionType = FRF_FREQUENCY_DISTRIBUTION_
TYPE.Logarithmic;
input.FrequencyHzResolutionForLinear = 0.1;

double duration = Ch.FRFCalculateDuration(input);
```

3.44.4 FRFReadServoParameters

Description

This function reads all required servo parameters for calculation of: Controller, OpenLoop, ClosedLoop etc.

Syntax

public ServoParameters object.FRFReadServoParameters(Axis axis);

Arguments

Axis	axis for parameter retrieval

Return Value

ServoParameters with axis values

Example

```
Api Ch = new Api();
Ch.OpenCommEthernet("10.0.0.100", 701);
int axis = 0;
ServoParameters servoParameters = Ch.FRFReadServoParameters((Axis)axis);
```

3.44.5 FRFCalculateControllerFRD

Description

The function calculates controller FRD (Frequency Response Data) based on servo parameters, servo loop and frequency vector

Syntax

public FRD object.FRFCalculateControllerFRD(ServoParameters servoParameters, FRF_LOOP_TYPE loopType, double[] frequencyHz);

Arguments

SERVO_PARAMETERS	Servo parameters for axis
FRF_LOOP_TYPE	Loop type to calculate
double[]	Input Frequencies for calculation in Hz

Return Value

FRD structure with calculated values

Example

```
Api Ch = new Api();
Ch.OpenCommEthernet("10.0.0.100", 701);
double[] frequenciesHz = new double[] { 100, 200, 300 };
ServoParameters servoParameters = Ch.FRFReadServoParameters((Axis)0);
FRD controllerFRD = Ch.FRFCalculateControllerFRD(servoParameters, FRF_
LOOP_TYPE.PositionVelocity, frequenciesHz);
```

3.44.6 FRFCalculateOpenLoopFRD

Description

This function calculates open loop FRD based on servo parameters, servo loop and frequency vector.

Syntax

public FRD object.FRFCalculateOpenLoopFRD(ServoParameters servoParameters, FRF_LOOP_TYPE openType, FRD plant);

Arguments

ServoParameters	Servo Parameters
FRF_LOOP_TYPE	Open Loop Type
FRD	plant FRD information

Return Value

FRD structure with calculated values

Example

```
Example:
// Example shows how open loop may be calculated based on measured plant
and controller with
// adjusted parameters

Api Ch = new Api();
Ch.OpenCommEthernet("10.0.0.100", 701);
FRFInput input = new FRFInput();
//Assign proper input parameters
FRFOutput output = Ch.FRFMeasure(input);
ServoParameters servoParameters = Ch.FRFReadServoParameters((Axis)0);
double origSLVKP = servoParameters.SLVKP;
servoParameters.SLVKP = origSLVKP*1.1;
FRD openLoop = Ch.FRFCalculateOpenLoopFRD(servoParameters,
input.LoopType, output.Plant);
```

3.44.7 FRFCalculateClosedLoopFRD

```
// Example shows how closed loop may be calculated based on measured
plant and controller with
// adjusted parameters

Api Ch = new Api();
Ch.OpenCommEthernet("10.0.0.100", 701);
FRFInput input = new FRFInput();
//Assign proper valuer for “input”
FRFOutput output = Ch.FRFMeasure(input);
ServoParameters servoParameters = Ch.FRFReadServoParameters((Axis)0);
double origSLVKP = servoParameters.SLVKP;
servoParameters.SLVKP = origSLVKP*1.1;
FRD closedLoop =
  Ch.FRFCalculateClosedLoopFRD(servoParameters, input.LoopType,
output.Plant);
```

3.44.8 FRFCalculateStabilityMargins

Description

The function calculates required stability margins based on the frequency response data of the open loop.

Syntax

public FRFStabilityMargins object.FRFCalculateStabilityMargins(FRD openLoop);

Arguments

FRD	open loop FRD parameters

Return Value

FRFStabilityMargins object with calculation results

Example

```
Api Ch = new Api();
Ch.OpenCommEthernet("10.0.0.100", 701);
FRFInput input = new FRFInput();
//Assign proper valuer for "input"
FRFOutput output = Ch.FRFMeasure(input);
ServoParameters servoParameters = Ch.FRFReadServoParameters((Axis)0);
double origSLVKP = servoParameters.SLVKP;
servoParameters.SLVKP = origSLVKP * 1.1;
FRD openLoop = Ch.FRFCalculateOpenLoopFRD(servoParameters,
input.LoopType, output.Plant);
FRFStabilityMargins stabilityMargins =
   Ch.FRFCalculateStabilityMargins(openLoop);
```

3.44.9 CalculateFFT

Description

This function calculates a Fast Fourier Transform.

Syntax

public void object.CalculateFFT(double[] @in, double[] outReal, double[] outImag);

Arguments

double[]	Array of input for FFT
double[]	real part of calculated FFT
double[]	imaginary part of calculated FFT

Return Value

None

3.44.10 RunJitterAnalysis

Description

Function executes jitter analysis.

Syntax

public JitterAnalysisOutput object.RunJitterAnalysis(JitterAnalysisInput analysisInput);

Arguments

JITTER_ANALYSIS_INPUT	Data for analysis

Return Value

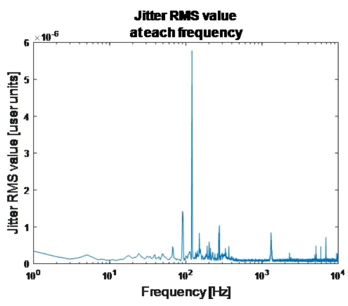
JitterAnalysisOutput data object

Example

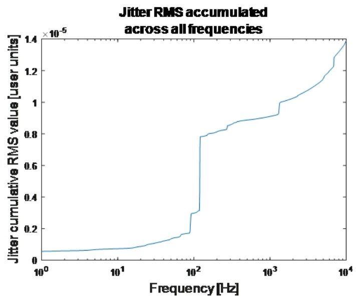
```
JitterAnalysisInput jitterInput = new JitterAnalysisInput();
int axis = 0;

double sampleDuration = 5;
string dataCollectionArray = "dataCollectionPE(" + sampleDuration * 20000
+ ")";
Ch.DeclareVariable(AcsplVariableType.ACSC_REAL_TYPE,
dataCollectionArray);
double EFAC = (double)Ch.ReadVariable("EFAC", ProgramBuffer.ACSC_NONE,
axis, axis, -1, -1);
int nmumberOfSamples = (int)(sampleDuration * 20000);
string transaction = "SPDC/r dataCollectionPE, 100000, 1/20000, 0, getspa
(0,\"axes[0].PE\")";
Ch.Transaction(transaction);
Thread.Sleep((int)(sampleDuration*1000));
double[] jitter = (double[])Ch.ReadVariable("dataCollectionPE",
ProgramBuffer.ACSC_NONE, 0, nmumberOfSamples - 1, -1, -1);

jitterInput.DesiredFrequencyResolutionHz = 1;
jitterInput.FrequencyBandsHz = new double[6] { 0, 100, 150, 300, 500,
1000 };
jitterInput.Jitter = new double[jitter.Length];
for (int k = 0; k < nmumberOfSamples; k++)
jitterInput.Jitter[k] = jitter[k] * EFAC;


jitterInput.JitterFrequencyBandsCumulativeAmplitudeRMSthresholds = new
double[5] { 1e-6, 2e-6, 1e-6, 2e-6, 1e-6 };
jitterInput.SamplingFrequencyHz = 20000;
jitterInput.WindowType = FRF_WINDOW_TYPE.Hanning;
JitterAnalysisOutput jitterOutput = Ch.RunJitterAnalysis(jitterInput);

jitterInput->desiredFrequencyResolutionHz = 1;
```

```
jitterInput->frequencyBandsHz = new double[6] {0, 100, 150, 300, 500,
1000};
jitterInput->frequencyBandsHzLength = 6;
jitterInput->jitter = new double[nmumberOfSamples];
for (int k = 0; k < nmumberOfSamples; k++)
       jitterInput->jitter[k] = jitter[k] * EFAC;

jitterInput->jitterLength = nmumberOfSamples;
jitterInput->jitterFrequencyBandsCumulativeAmplitudeRMSthreshold = new
double[5] {1e-6, 2e-6, 1e-6, 2e-6, 1e-6};
jitterInput->samplingFrequencyHz = 20000;
jitterInput->windowType = FRF_WINDOW_TYPE::Hanning;
int errorCode = 0;
acsc_JitterAnalysis(jitterInput, &jitterOutput, &errorCode);
```

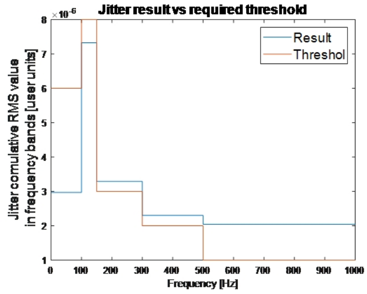
Analysis Results



>



>



3.44.11 FRFCrossCouplingMeasure

Description

This function returns the results of the cross-coupling measurement calculation.

Syntax

```
FRFCrossCouplingOutput FRFCrossCouplingMeasure(FRFCrossCouplingInput
input);
```

Example

```
FRFCrossCouplingInput Input = new FRFCrossCouplingInput();
Input.Axes = new int[] { 0, 6 };
Input.ExcitationAmplitudePercentIp = new double[] { 15, 15 };
Input.StartFreqHz = 10;
Input.EndFreqHz = 1000;
Input.DurationSec = 0.25;
Input.NumberOfRepetitions = 2;
Input.FreqPerDec = 50;
Input.FrequencyDistributionType = FRF_FREQUENCY_DISTRIBUTION_
TYPE.Logarithmic;
```

```
api.Enable((Axis)Input.Axes[0]);
api.Enable((Axis)Input.Axes[1]);
api.WaitMotorEnabled((Axis)Input.Axes[0], 1, 5000);
api.WaitMotorEnabled((Axis)Input.Axes[1], 1, 5000);
var outputCC = api.FRFCrossCouplingMeasure(Input);
```

3.45 FRF Data Structures

3.45.1 FRFInput

Description

The object defines the input for the FRF excitation signal.

Syntax

FRFInput
{
int axis;
FRF_LOOP_TYPE loopType ;
FRF_EXCITATION_TYPE excitationType;
FRF_CHIRP_TYPE chirpType;
FRF_WINDOW_TYPE windowType ;
overlap FRF_OVERLAP overlap ;
FRF_FREQUENCY_DISTRIBUTION_TYPE FrequencyDistributionType;
double startFreqHz;
double endFreqHz;
int freqPerDec;
double highResolutionStart;
int highResolutionFreqPerDec;
double excitationAmplitudePercentIp;
double durationSec;
int numberOfRepetitions;
double[] userDefinedExcitationSignal
int userDefinedExcitationSignalLength;
double[] inRaw;
double[] outRaw;
int lengthRaw;
bool recalculate;
};

Arguments

int	axis	Axis where excitation signal is injected
FRF_LOOP_TYPE	LoopType	ACS specific definition of servo loop that defines the type of Plant that will be measured

FRF_ EXCITATION_ TYPE	ExcitationType	Defines the excitation signal type **White noise**: generated for duration specified by **durationSec** and **numberOfRepetitions**. Signal is fully uncorrelated (not pseudo-random noise). Standard deviation is determined by the **excitationAmplitudePercentIp** . **ChirpPeriodic**: frequency range defined by the startFreqHz and endFreqHz is covered during the time period defined by the durationSec. The signal is repeated as defined in **numberOfRepetitions**. **UserDefined:** The signal is repeated as defined in **numberOfRepetitions**.
FRF_CHIRP_TYPE	ChirpType	Defines the type of Chirp signal. Applicable only if excitationType is *ChirpPeriodic*
FRF_WINDOW_ TYPE	WindowType	Defines the window type for windowing the signals before computing the FFT
FRF_OVERLAP	Overlap	Defines the amount of signals overlap for Welch averaging

FRF_ FREQUENCY_ DISTRIBUTION_ TYPE	DistributionType	Describes distribution of points of the resulted Plant. Has two options: Logarithmic: in this case user may specify **startFreqHz**, **endFreqHz**, **freqPerDec**, **HighResolutionStart** and **highResolutionFreqPerDec**. If **HighResolutionStart** falls between **startFreqHz** and **endFreqHz** and **highResolutionFreqPerDec** > **freqPerDec** then frequency range will have two regions with different frequency densities. Otherwise only the **startFreqHz**, **endFreqHz**, **freqPerDec** will determine the frequency range and density. Linear: frequency range is determined by the **startFreqHz** and **endFreqHz**, while frequency resolution is by the **durationSec** parameter.
double	startFreqHz	Defines start frequency of the resulted plant in Hz
double	endFreqHz	Defines endfrequency of the resulted plant in Hz
int	freqPerDec	Defines number of points per decade for standard resolution
double	highResolutionStart	Frequency where high resolution starts. High resolution feature is active only if highResolutionStart is within (startFreqHz-endFreqHz) range
int	highResolutionFreqPerDec	Defines number of frequencies per decade for high resolution range. High resolution feature is active only if highResolutionStart is within (startFreqHz-endFreqHz) range

double	excitationAmplitudePercentIp	Excitation amplitude of the signal defined in % of the drives peak current. In case of white noise this parameter determines the standard deviation of the noise.
double	durationSec	Duration of a single excitation in seconds. For periodic chirp it is the time required to go through all frequencies once. This parameter is ignored if selected excitationType = UserDefined
int	NumberOfRepititions	Number of repetitions of the excitation signal. In case of **excitationType** = ChirpPeriodic or UserDefined the signal is repeated according to **numberOfRepetitions** value. In case of white noise overall excitation duration will be excitation duration multiplied by the **numberOfRepetitions**.
double	UserDefinedExcitationSignal	Pointer to user defined excitation signal
int	UserDefinedExcitationSignalLength	Length of the user defined excitation signal.
double	InRaw	Raw excitation signal as measured during FRF calculation. Used only if **recalculate** = true
double	OutRaw	Raw measures signal as measured during FRF calculation. Used only if **recalculate** = true
bool	Recalculate	Recalculates the FRF instead of measuring it again.

3.45.2 FRFOutput

Description

The structure holds data returned by the FRF.

Syntax

```
FRFOutput
{
FRD Plant;
FRD PlantVelocityToPosition;
FRD CoherenceVelocityToPosition;
FRD Controller;
FRD OpenLoop;
FRD ClosedLoop;
FRD Sensitivity;
FRD Coherence;
FRF_LOOP_TYPE loopType;
FRF_STABILITY_MARGINS stabilityMargins;
double excitationAmplitude;
double[] InRaw;
double[] OutRaw;
int lengthRaw;
};
```

Arguments

FRD	Plant	Measured frequency response data (FRD) of the Plant
FRD	PlantVelocityToPosition	Transfer function from the velocity signal to position signal
FRD	CoherenceVelocityToPosition	Coherence of the measurement of the velocity to plant measurement
FRD	Controller	Calculated FRD of the controller. Calculation is based on standard servo parameters available in SERVO_PARAMETERS structure
FRD	openLoop	Calculated FRD of the openLoop based on plant and controller FRDs
FRD	ClosedLoop	Calculated FRD of the closedLoop based on plant and controller FRD’s. Closed loop will take into account SLAFF influence when it is relevant
FRD	Sensitivity	Calculated FRD of the sensitivity based on plant and controller FRD’s
FRD	Coherence	Measured FRD of the coherence amplitude. Imaginary part of the coherence amplitude is 0

FRF_LOOP_TYPE	LoopType	ACS specific definition of servo loop that defines the type of Plant that will be measured
FRF_STABILITY_MARGINS	StabilityMargins	Stability margins of the resulting closed loop. In case of the openLoop measurement stability margins are calculated for **PositionVelocity** loop
double	ExcitationAmplitude	Excitation amplitude used during measurement
double[]	InRaw	Raw excitation signal as measured during FRF calculation. Used only if **recalculate** = true
double[]	OutRaw	Raw measured signal as measured during FRF calculation. Used only if **recalculate** = true

3.45.3 FRF_DURATION_CALCULATION_PARAMETERS

Description

The structure contains the results of the duration calculation.

Syntax

```
FRF_DURATION_CALCULATION_PARAMETERS
{
 double startFreqHz;
double endFreqHz;
int freqPerDec;
double highResolutionStart;
int highResolutionFreqPerDec;
FRF_FREQUENCY_DISTRIBUTION_TYPE FrequencyDistributionType;
int frequencyHzResolutionForLinear;
}
```

Arguments

double	startFreqHz	Defines start frequency of the resulted plant in Hz *Ignored if frequency distribution type is set to Linear
double	endFreqHz	Defines end frequency of the resulted plant in Hz *Ignored if frequency distribution type is set to Linear
int	freqPerDec	Defines number of points per decade for standard resolution *Ignored if frequency distribution type is set to Linear
double	highResolutionStart	Frequency where high resolution starts. High resolution feature is active only if **highResolutionStart** is within (**startFreqHz**-**endFreqHz**) range *Ignored if frequency distribution type is set to Linear
int	highResolutionFreqPerDec	Defines number of frequencies per decade for high resolution range. High resolution feature is active only if **highResolutionStart** is within (**startFreqHz**-**endFreqHz**) range *Ignored if frequency distribution type is set to Linear
FRF_FREQUENCY_DISTRIBUTION_TYPE	FrequencyDistributionType	Describes distribution of points of the resulting plant.

int	frequencyHzResolutionForLinear	Defines required frequency resolution if frequency distribution type is set to Linear, otherwise ignored.

3.45.4 FRD

Description

The structure holds FRD data.

Syntax

```
struct
{
 double[] real;
double[] imag;
double[] frequencyHz;
unsigned int length;
};
```

Arguments

double	Real	Real part of FRD
double	Imag	Imaginary part of FRD
double	FrequencyHz	Frequencies in Hz units

3.45.5 FRF_STABILITY_MARGINS

Description

The structure holds stability margins.

Syntax

```
struct
{
 double[] GainMarginArray;
double[] GainMarginArrayFrequencyHz;
double GainMarginWorst;
double GainMarginWorstFrequencyHz;
double[] PhaseMarginArray;
double[] PhaseMarginArrayFrequencyHz;
double PhaseMarginWorst;
double PhaseMarginWorstFrequencyHz;
```

```
double ModulusMargin;
double ModulusMarginFrequencyHz;
double Bandwidth;
};
```

Arguments

double	GainMarginArray	Array of gain margin values
double	GainMarginArrayFrequencyHz	Array of gain margin frequency values in Hz.
double	GainMarginWorst	Most critical gain margin
double	GainMarginWorstFrequencyHz	Most critical stability margin frequency in Hz.
double	PhaseMarginArray	Array of phase margin values
double	PhaseMarginArrayFrequencyHz	Array of phase margin frequency values in Hz.
double	PhaseMarginWorst	Most critical phase margin
double	PhaseMarginWorstFrequencyHz	Most critical phase margin frequency in Hz.
double	ModulusMargin	Modulus margin
double	ModulusMarginFrequencyHz	Modulus margin frequency in Hz.
double	Bandwidth	Bandwidth Hz. Defined as first 0dB cross of the open loop

3.45.6 JITTER_ANALYSIS_INPUT

Description

The structure holds input values for jitter analysis.

Syntax

```
struct
{
 double[] Jitter;
int SamplingFrequencyHz;
double DesiredFrequencyResolutionHz;
double[] FrequencyBandsHz;
double[] JitterFrequencyBandsCumulativeAmplitudeRMSthreshold;
FRF_WINDOW_TYPE windowType;
};
```

Arguments

double	Jitter	Jitter values. In most cases position error should be used for the analysis.
int	SamplingFrequencyHz	Sampling frequency of jitter signal
double	DesiredFrequencyResolutionHz	Desired frequency resolution of the jitter analysis result in frequency domain
double	FrequencyBandsHz	Frequency bands for jitter analysis For example, 0,1000,2000,5000 defines three frequency bands 1. 0-1000Hz 2. 1000-2000Hz 3. 2000-5000Hz
double	JitterFrequencyBandsCumulativeAmplitudeRMSthreshold	Threshold for jitter level for frequency bands defined by **frequencyBandsHz** parameter. The length of this parameter should be **frequencyBandsHzLength** - 1
FRF_WINDOW_TYPE	WindowType	Defines the window type for windowing the signals before computing the FFT

3.45.7 JITTER_ANALYSIS_OUTPUT

Description

The structure holds the results of the jitter analysis.

Syntax

```
struct
{
 double[] JitterAmplitudeRMS;
double[] JitterCumulativeAmplitudeRMS;
double[] FrequencyHz;
double[] JitterFrequencyBandsCumulativeAmplitudeRMS;
double[] FrequencyBandsHz;
int JitterFrequencyBandsResultBool;
double JitterRMS;
double JitterAmplitudePeak2Peak;
} ;
```

Arguments

double	JitterAmplitudeRMS	Jitter values in frequency domain representing rms value of the jitter at unit of the Jitter variable in Jitter analysis input.
double	JitterCumulativeAmplitudeRMS	Cumulative jitter rms values as a function of frequencies. Rapid change in this graph implies a problematic area in frequency domain
double	FrequencyHz	Frequency array for **jitterAmplitudeRMS** and **jitterCumulativeAmplitudeRMS**
double	JitterFrequencyBandsCumulativeAmplitudeRMS	Jitter RMS values in frequency bands defined by the **frequencyBandsHz** parameter of the jitter analysis input parameter The length of this array is frequencyBandsHzLength - 1
double	FrequencyBandsHz	Frequency bands as defined in the **frequencyBandsHz** parameter of the jitter analysis input parameter
int	jitterFrequencyBandsResultBool	true if all values in **jitterFrequencyBandsCumulativeAmplitudeRMS** are below **JitterFrequencyBandsCumulativeAmplitudeRMSthreshold**

double	JitterRMS	RMS value of jitter in time domain
double	JitterAmplitudePeak2Peak	Peak to peak value of jitter in time domain

3.45.8 SERVO_PARAMETERS

Description

The structure holds the axis servo parameters read from the controller.

Syntax

```
struct
{
 double SLIKP;
 double SLIKP;
 double SLILI;
 double SLPKP;
 double SLPKI;
 double SLPLI;
 double SLVKP;
 double SLVKI;
 double SLVLI;
 double SLVSOF;
 double SLVSOFD;
 double SLVNFRQ;
 double SLVNWID;
 double SLVNATT;
 double SLVB0NF;
 double SLVB0DF;
 double SLVB0ND;
 double SLVB0DD;
 double SLVB1NF;
 double SLVB1DF;
 double SLVB1ND;
 double SLVB1DD;
 double XVEL;
 double EFAC;
 double SLVRAT;
 double SLAFF;
 INT32 MFLAGS;
 INT32 MFLAGSX;
} ;
```

Arguments

double	SLIKP	value read from controller
double	SLIKP	value read from controller
double	SLILI	value read from controller
double	SLPKP	value read from controller
double	SLPKP	value read from controller
double	SLPKI	value read from controller
double	SLPLI	value read from controller
double	SLVKP	value read from controller
double	SLVLI	value read from controller
double	SLVSOF	value read from controller
double	SLVSOFD	value read from controller
double	SLVNFRQ	value read from controller
double	SLVNWID	value read from controller
double	SLVNATT	value read from controller
double	SLVB0NF	value read from controller
double	SLVB0DF	value read from controller
double	SLVB0ND	value read from controller
double	SLVB0DD	value read from controller
double	SLVB1NF	value read from controller
double	SLVB1DF	value read from controller
double	SLVB1ND	value read from controller
double	SLVB1DD	value read from controller
double	XVEL	value read from controller
double	EFAC	value read from controller
double	SLVRAT	value read from controller
double	SLAFF	value read from controller

INT32	MFLAGS	value read from controller
INT32	MFLAGSX	value read from controller

3.45.9 FRFCrossCouplingInput

Description

Sets the input parameters for measurement of cross-coupling effects.

Name	Type	Default and range	Description
Axes	Int []	Not defined	Axes involved in the cross coupling measurement.
Not available	Int	0-127	Length of Axes array. Determines the number of involved axes.
CrossCouplingType	FRF_CROSS_ COUPLING_TYPE	Complete CompleteOpen	Complete – measures the cross copling between all axes in closed loop. CompleteOpen – measures the cross copling between all axes in open loop.

Name	Type	Default and range	Description
ExcitationType	FRF_EXCITATION_ TYPE	WhiteNoise ChirpPeriodic UserDefined	Defines the excitation signal type White noise: generated for duration specified by durationSec and numberOfRepetitions. Signal is fully uncorrelated (not pseudo random noise). Standard deviation is determined by the excitationAmplitudePercentIp ChirpPeriodic: frequency range defined by the startFreqHz and endFreqHz is covered during the time period defined by the durationSec. The signal is repeated as defined in numberOfRepetitions. UserDefined: The signal is repeated as defined in numberOfRepetitions.
ChirpType	FRF_CHIRP_TYPE	LogarithmicChirp LinearChirp	Defines the type of Chirp signal. Applicable only if excitationType is ChirpPeriodic
WindowType	FRF_WINDOW_ TYPE	Hanning Hamming Rectangular	Defines the window type for filtering the signals before computing the FFT
Overlap	FRF_OVERLAP	NoOverlap HalfSignal	Defines the amount of signals overlap for Welch averaging

Name	Type	Default and range	Description
FrequencyDistributionType	FRF_FREQUENCY_ DISTRIBUTION_ TYPE	Logarithmic Linear	Describes distribution of points of the resulted Plant. Has two options Logarithmic: in this case user may specify startFreqHz, endFreqHz, freqPerDec, HighResolutionStart and highResolutionFreqPerDec. If HighResolutionStart falls between startFreqHz and endFreqHz and highResolutionFreqPerDec > freqPerDec then frequency range will have two regions with different frequency densities. Otherwise only the startFreqHz, endFreqHz, freqPerDec will determine the frequency range and density. Linear: frequency range is determined by the startFreqHz and endFreqHz, while frequency resolution is by duration of the single measurement
StartFreqHz	double	30, 1-5000	Defines start frequency of the resulted plant in Hz
EndFreqHz	double	3000, 1-5000	Defines end frequency of the result plant in Hz
FreqPerDec	int	50, 10-1000	Defines number of points per decade for standard resolution

Name	Type	Default and range	Description
HighResolutionStart	double	500	Frequency where high resolution starts. High resolution feature is active only if highResolutionStart is within (startFreqHz-endFreqHz) range
HighResolutionFreqPerDec	int	500	Defines number of frequencies per decade for high resolution range. High resolution feature is active only if highResolutionStart is within (startFreqHz-endFreqHz) range
ExcitationAmplitudePercentIp	double[]	1, 1e-4 – 50	Excitation amplitude of the signal defined in % of the drives peak current. It is defined for each axis involved in the cross coupling measurement. In case of white noise this parameter determines the standard deviation of the noise.
DurationSec	double	5, 1e6-100	Duration of a single excitation in seconds. For periodic chirp it is the time that take to go through all frequencies once. This parameters ignored if selected excitationType = UserDefined.
NumberOfRepetitions	int	1-100	Number of repetitions of the excitation signal. In case of excitationType = ChirpPeriodic or UserDefined the signal is repeated according to numberOfRepetitions value. In case of white noise overall excitation duration will be excitation duration multiplied by the numberOfRepetitions.

3.45.10 FRFCrossCouplingOutput

Description

Reads the results of measurement of cross-coupling effects for use in FRF calculations.

Name	Type	Description
Plant	FRD[]	Measured frequency response data (FRD) of the Plant matrix
Controller	FRD[]	Calculated FRD of the controller matrix. Calculation is based on standard servo parameters available in SERVO_ PARAMETERS structure. Only the main diagonal has valid controller frequency response. Rest is euther zero or NULL.
characteristicPolynomial	FRD	Calculated FRD of the characteristic polynomial based on plant and controller FRD's
ClosedLoop	FRD[]	Calculated FRD of the ClosedLoop matrix based on plant and controller FRD's.
Sensitivity	FRD[]	Measured FRD of the sensitivity matrix.
CoherencePS	FRD[]	Measured FRD of the coherence amplitude of the precess sensitivity matrix. Imaginary part of the coherence amplitude is 0
CoherencePS	FRD[]	Measured FRD of the coherence amplitude of the sensitivity matrix. Imaginary part of the coherence amplitude is 0
RGA	FRD[]	Relative gain array matrix
crossCouplingType	FRF_LOOP_TYPE	Determines the cross coupling measurement type
StabilityMargins	FRFStabilityMargins	Stability margins calculated based on characteristic polynomial frequency respose
ExcitationAmplitude	double[]	Excitation amplitude used during measurement

3.46 Controller Protection Functions

3.46.1 DefineControllerProtection

Description

The function applies protection for a controller, reboots the controller and waits for process completion.

Syntax

Object.DefineControllerProtection(ControllerProtectionFlags flags, ProgramBuffer[] noEditBuffers, ProgramBuffer [] noViewBuffers, string standardVariables, string password, int timeout)

Async Syntax

ACSC_WAITBLOCK Object.DefineControllerProtection(ControllerProtectionFlags flags, ProgramBuffer[] noEditBuffers, ProgramBuffer[] noViewBuffers, string standardVariables, string password, int timeout)

Arguments

Flags	ACSC_PRIMARY_PROTECTION - defines primary protection. ACSC_SECONDARY_PROTECTION - defines secondary protection. ACSC_ALLOW_SYSTEM_RECONFIGURATION - Allows system reconfiguration. If one of the protections does not allow system reconfiguration of the system, then the other cannot allow it either. ACSC_VARIABLES_PROTECTION – define variable protection. (For primary protection only). ACSC_PRIMARY_PROTECTION and ACSC_SECONDARY_PROTECTION flags must not be specified together.
NoEditBuffers	[Optional] Array of buffer constants. Each element specifies one involved buffer: ACSC_BUFFER_0 corresponds to buffer 0, ACSC_BUFFER_1 to buffer 1, etc. After the last buffer, one additional element must be located that contains -1 which marks the end of the array. Set this argument to ACSC_NONE if not used.
NoViewBuffers	[Optional] Array of buffer constants. Each element specifies one involved buffer: ACSC_ BUFFER_0 corresponds to buffer 0, ACSC_BUFFER_1 to buffer 1, etc. After the last buffer, one additional element must be located that contains -1 which marks the end of the array. Set this argument to ACSC_NONE if not used.

StandardVariables	[Optional] String of variables to be protected. Only with primary protection. The string contains chained variables names separated by '\r'(13) character. Set this argument to NULL if not used.
Password	Password (must be 4 or more characters).
Timeout	Maximum waiting time in milliseconds. As part of the process, controller will be rebooted. Recommended value 150000. Depends on network configuration.

Return Value

None

Remarks

Application and removal of controller protection involves writing to flash memory. Exceeding the rated number of flash memory writes (about 100,000) risks damage to the memory device.

Example

```
Api _API = new Api(); // declare it as global
ProgramBuffer[] noEdit = { ProgramBuffer.ACSC_BUFFER_0, ProgramBuffer.ACSC_NONE };
ProgramBuffer[] noView = { ProgramBuffer.ACSC_BUFFER_1, ProgramBuffer.ACSC_NONE };
string str = "VEL\rDEC\r";

_API.DefineControllerProtection(ControllerProtectionFlags.ACSC_PRIMARY_PROTECTION
| ControllerProtectionFlags.ACSC_ALLOW_SYSTEM_RECONFIGURATION |
ControllerProtectionFlags.ACSC_VARIABLES_PROTECTION, noEdit, noView, str, "1234",
150000);
```

3.46.2 RemoveControllerProtection

Description

The function removes protection, reboots the controller, and waits for process completion.

Syntax

Object.RemoveControllerProtection(ControllerProtectionFlags flags, string password, int timeout)

Async Syntax

ACSC_WAITBLOCK Object.RemoveControllerProtection(ControllerProtectionFlags flags, string password, int timeout)

Arguments

Flags	ACSC_PRIMARY_PROTECTION - Remove primary protection ACSC_SECONDARY_PROTECTION - Remove secondary protection
Password	Password
Timeout	Maximum waiting time in milliseconds As part of the process, controller will be rebooted. Recommended value 150000. Depends on network configuration.

Return Value

None

Remarks

Application and removal of controller protection involves writing to flash memory. Exceeding the rated number of flash memory writes (about 100,000) risks damage to the memory device.

Example

```
Api _API = new Api(); // declare it as global
_API.RemoveControllerProtection(ControllerProtectionFlags.ACSC_PRIMARY_PROTECTION,
"1234", 90000);
```

3.46.3 TemporarilyDisableVariableProtection

Description

The function allows modification and saving of protected variables. Related only for primary protection.

Syntax

Object.TemporarilyDisableVariableProtection (string password)

Async Syntax

ACSC_WAITBLOCK Object.TemporarilyDisableVariableProtection (string password)

Arguments

Password	Primary protection password.

Return Value

None

Remarks

Application and removal of controller protection involves writing to flash memory. Exceeding the rated number of flash memory writes (about 100,000) risks damage to the memory device.

Example

```
Api _API = new Api(); // declare it as global
_API.TemporarilyDisableVariableProtection("1234");
```

3.46.4 RestoreVariableProtection

Description

The function restores variable protection that was temporarily disabled.

Syntax

Object.RestoreVariableProtection ()

Async Syntax

ACSC_WAITBLOCK Object.RestoreVariableProtection ()

Return Value

None

Remarks

Application and removal of controller protection involves writing to flash memory. Exceeding the rated number of flash memory writes (about 100,000) risks damage to the memory device.

Example

```
Api _API = new Api(); // declare it as global
_API.RestoreVariableProtection();
```

4. Enumerations

This chapter presents the built-in enumeration types that are used for establishing various properties during program runtime. For each property, the name of the constant, its value and a description are given.

4.1 General Definitions

Syntax:

public enum GeneralDefinition

Table 4-1. General Definitions

Name	Value	Description
ACSC_VER	0x63F1100	
ACSC_DC_VAR_MAX_CHANNEL	10	
ACSC_INT_BINARY	4	Integer type of the variable
ACSC_INVALID	(HANDLE)-1	
ACSC_NONE	-1	
ACSC_REAL_BINARY	8	Real type of the variable
ACSC_FILE	-2	
ACSC_DEFAULT_REMOTE_PORT	9999	
ACSC_AXES_MAX_NUMBER	128	Maximum number of axes
ACSC_BUFFERS_MAX_NUMBER	65	Maximum number of Program buffers
ACSC_SP_MAX_NUMBER	128	Maximum number of Servo Processors
ACSC_DC_VAR_MAX_NUMBER	10	
ACSC_MAX_LINE	100000	
ACSC_COUNTERCLOCKWISE	1	
ACSC_CLOCKWISE	-1	
ACSC_POSITIVE_DIRECTION	1	

Name	Value	Description
ACSC_NEGATIVE_DIRECTION	-1	

4.2 General Communication Options

Syntax:

public enum CommOptions

Table 4-2. General Communication Options

Name	Value	Description
ACSC_NONE	0	
ACSC_ALL	-1	
ACSC_COMM_USE_CHECKSUM	0x00000001	The communication mode when each command is sent to the controller with checksum and the controller also responds with checksum.
ACSC_COMM_AUTORECOVER_HW_ERROR	0x00000002	When a hardware error is detected in the communication channel and this bit is set, the library automatically repeats the transaction, without counting iterations. By default, this flag is not set.

4.3 Ethernet Communication Options

Syntax:

public enum EthernetCommOption

Table 4-3. Ethernet Communication Options

Name	Value	Description
ACSC_SOCKET_DGRAM_PORT	700	The library opens Ethernet communication using the connectionless socket and UDP communication protocol.
ACSC_SOCKET_STREAM_PORT	701	The library opens Ethernet communication using the connection- oriented socket and TCP communication protocol.

4.4 Serial Communication Options

Syntax:

public enum SerialCommOption

Table 4-4. Serial Communication Option

Name	Value	Description
ACSC_AUTO	-1	

4.5 Axis Definitions

Syntax:

public enum Axis

Description

The general format for any axis definition is:

ACSC_AXIS_index

Where index is a number that ranges between 0 and 127, such as **ACSC_AXIS_0**, **ACSC_AXIS_1**, **ACSC_AXIS_30**, etc. The axis constant contains the value associated with the index, that is, **ACSC_AXIS_0** has a value of 0, **ACSC_AXIS_1** has a value of 1, and so forth. **ACSC_PAR_ALL** stands for all parameters (system and all axes parameters) and **ACSC_SYSTEM** stands for the system parameters. **ACSC_NONE** has a value of -1 and serves as axes array terminator.

4.6 Buffer Definitions

Syntax:

public enum ProgramBuffer

Description

The general format for any buffer definition is:

ACSC_BUFFER_index

Where index is a number that ranges between 0 and 64, such as **ACSC_BUFFER_0**, **ACSC_BUFFER_1**, **ACSC_BUFFER_64**, etc. The buffer constant contains the value associated with the index, that is, **ACSC_BUFFER_0** has a value of 0, **ACSC_BUFFER_1** has a value of 1, and so forth. **ACSC_BUFFER_ALL** stands for all buffers. **ACSC_NONE** has a value of -1 and serves as buffer array terminator.

Table 4-5. ProgramBuffer Values

Name	Value	Description
ACSC_NONE	-1	
ACSC_BUFFER_ALL	-2	
ACSC_BUFFER_0	0	
ACSC_BUFFER_1	1	
ACSC_BUFFER_2	2	
ACSC_BUFFER_3	3	

Name	Value	Description
ACSC_BUFFER_4	4	
ACSC_BUFFER_5	5	
ACSC_BUFFER_6	6	
ACSC_BUFFER_7	7	
ACSC_BUFFER_8	8	
ACSC_BUFFER_9	9	
ACSC_BUFFER_10	10	
ACSC_BUFFER_11	11	
ACSC_BUFFER_12	12	
ACSC_BUFFER_13	13	
ACSC_BUFFER_14	14	
ACSC_BUFFER_15	15	
ACSC_BUFFER_16	16	
ACSC_BUFFER_17	17	
ACSC_BUFFER_18	18	
ACSC_BUFFER_19	19	
ACSC_BUFFER_20	20	
ACSC_BUFFER_21	21	
ACSC_BUFFER_22	22	
ACSC_BUFFER_23	23	
ACSC_BUFFER_24	24	
ACSC_BUFFER_25	25	
ACSC_BUFFER_26	26	
ACSC_BUFFER_27	27	

Name	Value	Description
ACSC_BUFFER_28	28	
ACSC_BUFFER_29	29	
ACSC_BUFFER_30	30	
ACSC_BUFFER_31	31	
ACSC_BUFFER_32	32	
ACSC_BUFFER_33	33	
ACSC_BUFFER_34	34	
ACSC_BUFFER_35	35	
ACSC_BUFFER_36	36	
ACSC_BUFFER_37	37	
ACSC_BUFFER_38	38	
ACSC_BUFFER_39	39	
ACSC_BUFFER_40	40	
ACSC_BUFFER_41	41	
ACSC_BUFFER_42	42	
ACSC_BUFFER_43	43	
ACSC_BUFFER_44	44	
ACSC_BUFFER_45	45	
ACSC_BUFFER_46	46	
ACSC_BUFFER_47	47	
ACSC_BUFFER_48	48	
ACSC_BUFFER_49	49	
ACSC_BUFFER_50	50	
ACSC_BUFFER_51	51	

Name	Value	Description
ACSC_BUFFER_52	52	
ACSC_BUFFER_53	53	
ACSC_BUFFER_54	53	
ACSC_BUFFER_55	55	
ACSC_BUFFER_56	56	
ACSC_BUFFER_57	57	
ACSC_BUFFER_58	58	
ACSC_BUFFER_59	59	
ACSC_BUFFER_60	60	
ACSC_BUFFER_61	61	
ACSC_BUFFER_62	62	
ACSC_BUFFER_63	63	
ACSC_BUFFER_64	64	

4.7 Servo Processor (SP) Definitions

Syntax:

public enum ServoProcessor

Description

The general format for any SP definition is:

ACSC_SP_index

Where index is a number that ranges between 0 and 63, such as **ACSC_SP_0**, **ACSC_SP_1**, **ACSC_SP_63**, etc. The SP constant contains the value associated with the index, that is, **ACSC_SP_0** has a value of 0, **ACSC_SP_1** has a value of 1, and so forth. **ACSC_SP_ALL** stands for all SPs. **ACSC_NONE** has a value of -1 and serves as array terminator.

Table 4-6. ServoProcessor Values

NAME	Value	Description
ACSC_NONE	-1	
ACSC_SP_ALL	-2	

NAME	Value	Description
ACSC_SP_0	0	
ACSC_SP_1	1	
ACSC_SP_2	2	
ACSC_SP_3	3	
ACSC_SP_4	4	
ACSC_SP_5	5	
ACSC_SP_6	6	
ACSC_SP_7	7	
ACSC_SP_8	8	
ACSC_SP_9	9	
ACSC_SP_10	10	
ACSC_SP_11	11	
ACSC_SP_12	12	
ACSC_SP_13	13	
ACSC_SP_14	14	
ACSC_SP_15	15	
ACSC_SP_16	16	
ACSC_SP_17	17	
ACSC_SP_18	18	
ACSC_SP_19	19	
ACSC_SP_20	20	
ACSC_SP_21	21	
ACSC_SP_22	22	
ACSC_SP_23	23	

NAME	Value	Description
ACSC_SP_24	24	
ACSC_SP_25	24	
ACSC_SP_26	26	
ACSC_SP_27	27	
ACSC_SP_28	28	
ACSC_SP_29	29	
ACSC_SP_30	30	
ACSC_SP_31	31	
ACSC_SP_32	32	
ACSC_SP_33	33	
ACSC_SP_34	34	
ACSC_SP_35	35	
ACSC_SP_36	36	
ACSC_SP_37	37	
ACSC_SP_38	38	
ACSC_SP_39	39	
ACSC_SP_40	40	
ACSC_SP_41	41	
ACSC_SP_42	42	
ACSC_SP_43	43	
ACSC_SP_44	44	
ACSC_SP_45	45	
ACSC_SP_46	46	
ACSC_SP_47	47	

NAME	Value	Description
ACSC_SP_48	48	
ACSC_SP_49	49	
ACSC_SP_50	50	
ACSC_SP_51	51	
ACSC_SP_52	52	
ACSC_SP_53	53	
ACSC_SP_54	54	
ACSC_SP_55	55	
ACSC_SP_56	56	
ACSC_SP_57	57	
ACSC_SP_58	58	
ACSC_SP_59	59	
ACSC_SP_60	60	
ACSC_SP_61	61	
ACSC_SP_62	62	
ACSC_SP_63	63	
ACSC_SP_64	64	
ACSC_SP_65	65	
ACSC_SP_66	66	
ACSC_SP_67	67	
ACSC_SP_68	68	
ACSC_SP_69	69	
ACSC_SP_70	70	
ACSC_SP_71	71	

NAME	Value	Description
ACSC_SP_72	72	
ACSC_SP_73	73	
ACSC_SP_74	74	
ACSC_SP_75	75	
ACSC_SP_76	76	
ACSC_SP_77	77	
ACSC_SP_78	78	
ACSC_SP_79	79	
ACSC_SP_80	80	
ACSC_SP_81	81	
ACSC_SP_82	82	
ACSC_SP_83	83	
ACSC_SP_84	84	
ACSC_SP_85	85	
ACSC_SP_86	86	
ACSC_SP_87	87	
ACSC_SP_88	88	
ACSC_SP_89	89	
ACSC_SP_90	90	
ACSC_SP_91	91	
ACSC_SP_92	92	
ACSC_SP_93	93	
ACSC_SP_94	94	
ACSC_SP_95	95	

NAME	Value	Description
ACSC_SP_96	96	
ACSC_SP_97	97	
ACSC_SP_98	98	
ACSC_SP_99	99	
ACSC_SP_100	100	
ACSC_SP_101	101	
ACSC_SP_102	102	
ACSC_SP_103	103	
ACSC_SP_104	104	
ACSC_SP_105	105	
ACSC_SP_106	106	
ACSC_SP_107	107	
ACSC_SP_108	108	
ACSC_SP_109	109	
ACSC_SP_110	110	
ACSC_SP_111	111	
ACSC_SP_112	112	
ACSC_SP_113	113	
ACSC_SP_114	114	
ACSC_SP_115	115	
ACSC_SP_116	116	
ACSC_SP_117	117	
ACSC_SP_118	118	
ACSC_SP_119	119	

NAME	Value	Description
ACSC_SP_120	120	
ACSC_SP_121	121	
ACSC_SP_122	122	
ACSC_SP_123	123	
ACSC_SP_124	124	
ACSC_SP_125	125	
ACSC_SP_126	126	
ACSC_SP_127	127	

4.8 EtherCAT Flags

Syntax:

public enum EtherCatFlags

Table 4-7. EtherCAT Flags

Name	Value	Description
ACSC_NONE	0	Disable all EtherCAT flags
ACSC_ALL	-1	Enable all EtherCAT flags
ACSC_ETHERCAT_1BYTE	0x00000001	
ACSC_ETHERCAT_2BYTES	0x00000002	
ACSC_ETHERCAT_4BYTES	0x00000004	
ACSC_ETHERCAT_FLOAT	0x00000008	
ACSC_ETHERCAT_NETWORK_0	0x00000010	Function refers to first EtherCAT network
ACSC_ETHERCAT_NETWORK_1	0x00000020	Function refers to second of two Dual EtherCAT networks
ACSC_BIT_OFFSET	0x00000040	

4.9 Motion Flags

Syntax:

public enum MotionFlags

Table 4-8. Motion Flags

Name	Value	Description
ACSC_NONE	0	Disables all motion flags
ACSC_ALL	-1	Enables all motion flags
ACSC_AMF_WAIT	0x00000001	The controller plans the motion but doesn't start it until the Go method is executed.
ACSC_AMF_RELATIVE	0x00000002	The value of the point coordinate is relative to the end point coordinate of the previous motion.
ACSC_AMF_VELOCITY	0x00000004	The motion uses the specified velocity instead of the default velocity.
ACSC_AMF_ENDVELOCITY	0x00000008	The motion comes to the end point with the specified velocity
ACSC_AMF_REQUIRED_ VELOCITY	0x00000008	Specifies a Required velocity in the current point.
ACSC_AMF_ FASTLOADINGPEG	0x00000008	Fast loading of Random PEG arrays is activated.
ACSC_AMF_POSITIONLOCK	0x00000010	The slaved motion uses position lock. If the flag is not specified, velocity lock is used.
ACSC_AMF_VELOCITYLOCK	0x00000020	The slaved motion uses velocity lock.
ACSC_AMF_CYCLIC	0x00000100	The motion uses the point sequence as a cyclic array: after positioning to the last point it does positioning to the first point and continues.
ACSC_AMF_CORNER	0x00000100	
ACSC_AMF_DYNAMIC_ ERROR_COMPENSATION1D	0x00000100	

Name	Value	Description
ACSC_AMF_VARTIME	0x00000200	The time interval between adjacent points of the spline (arbitrary path) motion is non-uniform and is specified along with an each added point. If the flag is not specified, the interval is uniform.
ACSC_AMF_TIME	0x00000200	Minimum travel time in seconds
ACSC_AMF_STALLED	0x00000200	
ACSC_AMF_CUBIC	0x00000400	Use a cubic interpolation between the specified points (third-order spline) for the spline (arbitrary path) motion. If the flag is not specified, linear interpolation is used (first-order spline). Currently third-order spline is not supported.
ACSC_AMF_2	0x00000800	Use 20 kHz motion mode
ACSC_AMF_EXTRAPOLATED	0x00001000	Segmented slaved motion: if a master value travels beyond the specified path, the last or the first segment is extrapolated.
ACSC_AMF_ENVELOPE	0x00001000	Wait for motion termination before executing next command
ACSC_AMF_ENDLESS	0x00001000	Supports endless incremental PEG
ACSC_AMF_AXISLIMIT	0x00002000	Enable velocity limitations under axis limits.
ACSC_AMF_NURBS_EXCEPTION_LENGTH	0x00002000	Specify NURBS exception length.
ACSC_AMF_LOCAL	0x00002000	Interpret entered coordinates according to the Local Coordinate System.

Name	Value	Description
ACSC_AMF_MAXIMUM	0x00004000	Multi-axis motion does not use the motion parameters from the leading axis but calculates the maximum allowed motion velocity, acceleration, deceleration and jerk of the involved axes.
ACSC_AMF_MAXIMUM_ARR_SIZE	0x00004000	Supports 1D/2D dynamic error compensation for Incremental PEG.
ACSC_AMF_BSEGTIME	0x00004000	Segment time
ACSC_AMF_MODULE	0x00004000	The Positions Arrays is loaded once, provides pulses every Modulo cycle
ACSC_AMF_SYNCHRONOUS	0x00008000	Position Event Generation (PEG): Start PEG synchronously with the motion sequence.
ACSC_AMF_BSEGJERK	0x00008000	Segment jerk time
ACSC_AMF_JUNCTIONVELOCITY	0x00010000	Decelerate to corner.
ACSC_AMF_ANGLE	0x00020000	Do not treat junction as a corner, if junction angle is less than or equal to the specified value in radians.
ACSC_AMF_ACCURATE	0x00020000	
ACSC_AMF_BSEGACC	0x00020000	Segment acceleration time.
ACSC_AMF_NURBS_EXCEPTION_ANGLE	0x00020000	Requires additional parameter that specifies maximum angle in a control point.
ACSC_AMF_USERVARIABLES	0x00040000	Synchronize user variables with segment execution.
ACSC_AMF_MIN_AXIS_DIRECTION	0x00040000	Signals that the MinDirDistance parameter indicates a value defining actual motion as opposed to jitter
ACSC_AMF_INVERT_OUTPUT	0x00080000	The PEG pulse output is inverted.

Name	Value	Description
ACSC_AMF_CURVEVELOCITY	0x00100000	Decelerate to curvature discontinuity point.
ACSC_AMF_DWELLTIME	0x00100000	Dwell time between segments.
ACSC_AMF_DUMMY	0x00100000	
ACSC_AMF_FIXED_TIME	0x00100000	Specifies the exact travel time for the motion in seconds
ACSC_AMF_DYNAMICLOADINGPEG	0x00100000	Dynamic loading of positions is implemented
ACSC_AMF_CORNERDEVIATION	0x00200000	Use a corner rounding option with the specified permitted deviation.
ACSC_AMF_NURBS_CONSIDER_ACC	0x00200000	Allow the MG to deviate from specified axes acceleration parameter during velocity profile generation
ACSC_AMF_CORNERRADIUS	0x00400000	Use a corner rounding option with the specified permitted curvature.
ACSC_AMF_CORNERLENGTH	0x00800000	Use automatic corner rounding option.
ACSC_AMF_CURVEAUTO	0x01000000	Automatic curve calculations
ACSC_AMF_DYNAMIC_ERROR_COMPENSATION2D	0x01000000	Supports 2D dynamic error compensation for Incremental PEG.
ACSC_AMF_EXT_LOOP	0x02000000	Use external loops at corners
ACSC_AMF_EXT_LOOP_SYNC	0x04000000	Defines output bit to support external loop synchronization
ACSC_AMF_DELAY_MOTION	0x08000000	Defines actual motor movement delay
ACSC_AMF_LCI_STATE	0x10000000	Requires additional parameter that specify LCI state.
ACSC_AMF_LOCALCS	0x20000000	Interpret entered coordinates according to the Local Coordinate System.

Name	Value	Description
ACSC_AMF_DYNAMIC_ ERROR_COMPENSATION3D	0x20000000	Supports 3D dynamic error compensation for Incremental PEG
ACSC_AMF_KNOT	0x40000000	Specify knot delta

Syntax:

public enum RotationDirection

Table 4-9. Rotation Direction Flags

Name	Value	Description
ACSC_COUNTERCLOCKWISE	1	Counter clockwise rotation
ACSC_CLOCKWISE	-1	Clockwise rotation

Syntax:

public enum GlobalDirection

Table 4-10. Global Direction Flags

Name	Value	Description
ACSC_POSITIVE_DIRECTION	1	Axis movement in positive direction
ACSC_NEGATIVE_DIRECTION	-1	Axis movement in negative direction

Syntax:

public enum CornerFlags

Table 4-11. CornerFlags

Name	Value	Description
ACSC_NONE	0	
ACSC_NONE	-1	
ACSC_AMF_CORNERDEVIATION	0x00200000	
ACSC_AMF_CORNERRADIUS	0x00400000	
ACSC_AMF_CORNERLENGTH	0x00800000	

Syntax:

public enum BsegFlags

Table 4-12. BSEG Flags

Name	Value	Description
ACSC_NONE	0	
ACSC_ALL	-1	
ACSC_AMF_DWELLTIME	0x00100000	
ACSC_AMF_BSEGTIME	0x00004000	
ACSC_AMF_BSEGACC	0x00020000	
ACSC_AMF_BSEGJERK	0x00008000	
ACSC_AMF_CURVEAUTO	0x01000000	

4.10 Dynamic Error Compensation Flags

Syntax:

public enum DynamicErrorCompensationFlags

Table 4-13. Dynamic Error Compensation Flags

Name	Value	Description
ACSC_NONE	0	
ACSC_ALL	-1	
ACSC_DECOMP_00	0x00000001	
ACSC_DECOMP_01	0x00000002	
ACSC_DECOMP_02	0x00000004	
ACSC_DECOMP_PREVENT_COMP_INDEX_MARK_PEG	0x00000008	
ACSC_DECOMP_REFERENCED_AXIS	0x00000010	
ACSC_DECOMP_ANALOG_INPUT	0x00000020	
ACSC_DECOMP_FIRST_ANALOG_INPUT	0x00000040	
ACSC_DECOMP_SECOND_ANALOG_INPUT	0x00000080	
ACSC_DECOMP_THIRD_ANALOG_INPUT	0x00000100	

4.11 Data Collection Flags

Syntax:

public enum DataCollectionFlags

Table 4-14. Data Collection Flags

Name	Value	Description
ACSC_NONE	0	Disables all data collections flags
ACSC_ALL	-1	Enables all data collection flags
ACSC_DCF_TEMPORAL	0x00000001	Temporal data collection. The sampling period is calculated automatically according to the collection time.
ACSC_DCF_CYCLIC	0x00000002	Cyclic data collection uses the collection array as a cyclic buffer and continues infinitely. When the array is full, each new sample overwrites the oldest sample in the array.
ACSC_DCF_SYNC	0x00000004	Starts data collection synchronously to a motion. Data collection started with the **ACSC_DCF_SYNC** flag is called axis data collection.
ACSC_DCF_WAIT	0x00000008	Creates synchronous data collection, but does not start until the Go method is called. This flag can only be used with the **ACSC_DCF_SYNC** flag.

4.12 Motor State Flags

Syntax:

public enum MotorStates

Table 4-15. Motor State Flags

Name	Value	Description
ACSC_NONE	0	Disables all motor state flags
ACSC_ALL	-1	Enables all motor state flags
ACSC_MST_ENABLE	0x00000001	Motor is enabled
ACSC_MST_INPOS	0x00000010	Motor has reached a target position.
ACSC_MST_MOVE	0x00000020	Motor is moving.
ACSC_MST_ACC	0x00000040	Motor is accelerating.

4.13 Axis State Flags

Syntax:

public enum AxisStates

Table 4-16. Axis State Flags

Name	Value	Description
ACSC_NONE	0	Disables all axis state flags
ACSC_ALL	-1	Enables all axis state flags
ACSC_AST_LEAD	0x00000001	Axis is leading in a group.
ACSC_AST_DC	0x00000002	Axis data collection is in progress.
ACSC_AST_PEG	0x00000004	PEG for the specified axis is in progress.
ACSC_AST_PEGREADY	0x00000010	
ACSC_AST_MOVE	0x00000020	Axis is moving.
ACSC_AST_ACC	0x00000040	Axis is accelerating.
ACSC_AST_SEGMENT	0x00000080	
ACSC_AST_VELLOCK	0x00000100	Slave motion for the specified axis is synchronized to master in velocity lock mode.
ACSC_AST_POSLOCK	0x00000200	Slave motion for the specified axis is synchronized to master in position lock mode.

Syntax:

public enum OldAxis

Name	Value	Description
ACSC_AXIS_X	0	
ACSC_AXIS_Y	1	
ACSC_AXIS_Z	2	
ACSC_AXIS_T	3	

Name	Value	Description
ACSC_AXIS_A	4	
ACSC_AXIS_B	5	
ACSC_AXIS_C	6	
ACSC_AXIS_D	7	

Syntax:

public enum Axis

NAME	Value	Description
ACSC_NONE	-1	
ACSC_PAR_ALL	-2	
ACSC_SYSTEM	-3	
ACSC_AXIS_0	0	
ACSC_AXIS_1	1	
ACSC_AXIS_2	2	
ACSC_AXIS_3	3	
ACSC_AXIS_4	4	
ACSC_AXIS_5	5	
ACSC_AXIS_6	6	
ACSC_AXIS_7	7	
ACSC_AXIS_8	8	
ACSC_AXIS_9	9	
ACSC_AXIS_10	10	
ACSC_AXIS_11	11	
ACSC_AXIS_12	12	
ACSC_AXIS_13	13	

NAME	Value	Description
ACSC_AXIS_14	15	
ACSC_AXIS_15	15	
ACSC_AXIS_16	16	
ACSC_AXIS_17	17	
ACSC_AXIS_18	18	
ACSC_AXIS_19	19	
ACSC_AXIS_20	20	
ACSC_AXIS_21	21	
ACSC_AXIS_22	22	
ACSC_AXIS_23	23	
ACSC_AXIS_24	24	
ACSC_AXIS_25	25	
ACSC_AXIS_26	26	
ACSC_AXIS_27	27	
ACSC_AXIS_28	28	
ACSC_AXIS_29	29	
ACSC_AXIS_30	30	
ACSC_AXIS_31	31	
ACSC_AXIS_32	32	
ACSC_AXIS_33	33	
ACSC_AXIS_34	34	
ACSC_AXIS_35	35	
ACSC_AXIS_36	36	
ACSC_AXIS_37	37	

NAME	Value	Description
ACSC_AXIS_38	38	
ACSC_AXIS_39	39	
ACSC_AXIS_40	40	
ACSC_AXIS_41	41	
ACSC_AXIS_42	42	
ACSC_AXIS_43	43	
ACSC_AXIS_44	44	
ACSC_AXIS_45	45	
ACSC_AXIS_46	46	
ACSC_AXIS_47	47	
ACSC_AXIS_48	48	
ACSC_AXIS_49	49	
ACSC_AXIS_50	50	
ACSC_AXIS_51	51	
ACSC_AXIS_52	52	
ACSC_AXIS_53	53	
ACSC_AXIS_54	54	
ACSC_AXIS_55	55	
ACSC_AXIS_56	56	
ACSC_AXIS_57	57	
ACSC_AXIS_58	58	
ACSC_AXIS_59	59	
ACSC_AXIS_60	60	
ACSC_AXIS_61	61	

NAME	Value	Description
ACSC_AXIS_62	62	
ACSC_AXIS_63	63	
ACSC_AXIS_64	64	
ACSC_AXIS_65	65	
ACSC_AXIS_66	66	
ACSC_AXIS_67	67	
ACSC_AXIS_68	68	
ACSC_AXIS_69	69	
ACSC_AXIS_70	70	
ACSC_AXIS_71	71	
ACSC_AXIS_72	72	
ACSC_AXIS_73	73	
ACSC_AXIS_74	74	
ACSC_AXIS_75	75	
ACSC_AXIS_76	76	
ACSC_AXIS_77	77	
ACSC_AXIS_78	78	
ACSC_AXIS_79	79	
ACSC_AXIS_80	80	
ACSC_AXIS_81	81	
ACSC_AXIS_82	82	
ACSC_AXIS_83	83	
ACSC_AXIS_84	84	
ACSC_AXIS_85	85	

NAME	Value	Description
ACSC_AXIS_86	86	
ACSC_AXIS_87	87	
ACSC_AXIS_88	88	
ACSC_AXIS_89	89	
ACSC_AXIS_90	90	
ACSC_AXIS_91	91	
ACSC_AXIS_92	92	
ACSC_AXIS_93	93	
ACSC_AXIS_94	94	
ACSC_AXIS_95	95	
ACSC_AXIS_96	96	
ACSC_AXIS_97	97	
ACSC_AXIS_98	98	
ACSC_AXIS_99	99	
ACSC_AXIS_100	100	
ACSC_AXIS_101	101	
ACSC_AXIS_102	102	
ACSC_AXIS_103	103	
ACSC_AXIS_104	104	
ACSC_AXIS_105	105	
ACSC_AXIS_106	106	
ACSC_AXIS_107	107	
ACSC_AXIS_108	108	
ACSC_AXIS_109	109	

NAME	Value	Description
ACSC_AXIS_110	110	
ACSC_AXIS_111	111	
ACSC_AXIS_112	112	
ACSC_AXIS_113	113	
ACSC_AXIS_114	114	
ACSC_AXIS_115	115	
ACSC_AXIS_116	116	
ACSC_AXIS_117	117	
ACSC_AXIS_118	118	
ACSC_AXIS_119	119	
ACSC_AXIS_120	120	
ACSC_AXIS_121	121	
ACSC_AXIS_122	122	
ACSC_AXIS_123	123	
ACSC_AXIS_124	124	
ACSC_AXIS_125	125	
ACSC_AXIS_126	126	
ACSC_AXIS_127	127	

4.14 Index and Mark State Flags

Syntax:

public enum IndexStates

Table 4-17. Index and Mark State Flags

Name	Value	Description
ACSC_NONE	0	Disables all index state flags
ACSC_ALL	-1	Enables all index state flags
ACSC_IST_IND	0x00000001	Primary encoder index of the specified axis is latched.
ACSC_IST_IND2	0x00000002	Secondary encoder index of the specified axis is latched.
ACSC_IST_MARK	0x00000004	MARK1 signal has been generated and position of the specified axis was latched.
ACSC_IST_MARK2	0x00000008	MARK2 signal has been generated and position of the specified axis was latched.

4.15 Program State Flags

Syntax:

public enum ProgramStates

Table 4-18. Program State Flags

Name	Value	Description
ACSC_NONE	0	Disables all program state flags
ACSC_ALL	-1	Enables all program state flags
ACSC_PST_COMPILED	0x00000001	Program in the specified buffer is compiled.
ACSC_PST_RUN	0x00000002	Program in the specified buffer is running.
ACSC_PST_SUSPEND	0x00000004	Program in the specified buffer is suspended after the step execution or due to breakpoint in debug mode.
ACSC_PST_DEBUG	0x00000020	Program in the specified buffer is executed in debug mode, i.e. breakpoints are active.
ACSC_PST_AUTO	0x00000080	Auto routine in the specified buffer is running.

4.16 Safety Control Masks

Syntax:

public enum SafetyControlMasks: uint

Table 4-19. Safety Control Masks

Name	Value	Description	Type
ACSC_NONE	0	Disables all safety mask flags	
ACSC_ALL	unchecked((uint)-1L)	Enables all safety mask flags	
ACSC_SAFETY_RL	0x00000001	Right Limit	Motor fault
ACSC_SAFETY_LL	0x00000002	Left Limit	Motor fault
ACSC_SAFETY_NETWORK	0x00000004	Network Error	Network fault
ACSC_SAFETY_HOT	0x00000010	Motor Overheat	Motor fault
ACSC_SAFETY_SRL	0x00000020	Software Right Limit	Motor fault
ACSC_SAFETY_SLL	0x00000040	Software Left Limit	Motor fault
ACSC_SAFETY_ENCNC	0x00000080	Primary Encoder Not Connected	Motor fault
ACSC_SAFETY_ENC2NC	0x00000100	Secondary Encoder Not Connected	Motor fault
ACSC_SAFETY_DRIVE	0x00000200	Driver Alarm	Motor fault
ACSC_SAFETY_ENC	0x00000400	Primary Encoder Error	Motor fault
ACSC_SAFETY_ENC2	0x00000800	Secondary Encoder Error	Motor fault
ACSC_SAFETY_PE	0x00001000	Position Error	Motor fault
ACSC_SAFETY_CPE	0x00002000	Critical Position Error	Motor fault
ACSC_SAFETY_VL	0x00004000	Velocity Limit	Motor fault
ACSC_SAFETY_AL	0x00008000	Acceleration Limit	Motor fault

Name	Value	Description	Type
ACSC_SAFETY_CL	0x00010000	Current Limit	Motor fault
ACSC_SAFETY_SP	0x00020000	Servo Processor Alarm	Motor fault
ACSC_SAFETY_STO	0x00040000	STO Alarm	
ACSC_SAFETY_HSSINC	0x00100000		
ACSC_SAFETY_EXTNT	0x00800000		
ACSC_SAFETY_TEMP	0x01000000		
ACSC_SAFETY_PROG	0x02000000	Program Error	System fault
ACSC_SAFETY_MEM	0x04000000	Memory Overuse	System fault
ACSC_SAFETY_TIME	0x08000000	Time Overuse	System fault
ACSC_SAFETY_ES	0x10000000	Emergency Stop	System fault
ACSC_SAFETY_INT	0x20000000	Servo Interrupt	System fault
ACSC_SAFETY_INTGR	0x40000000	Integrity Violation	System fault
ACSC_SAFETY_FAILURE	0x80000000		

See the *SPiiPlus ACSPL+ Programmer's Guide* for detailed explanation of faults.

4.17 Interrupt Types

Syntax:

public enum Interrupts

Table 4-20. Interrupt Types

Name	Value	Description
ACSC_INTR_PEG	3	
ACSC_INTR_MARK1	7	
ACSC_INTR_MARK2	8	
ACSC_INTR_EMERGENCY	15	EMERGENCY STOP signal has been generated.
ACSC_INTR_PHYSICAL_MOTION_END	16	Physical motion has finished.
ACSC_INTR_LOGICAL_MOTION_END	17	Logical motion has finished
ACSC_INTR_MOTION_FAILURE	18	
ACSC_INTR_MOTOR_FAILURE	19	Motor has been disabled due to a fault.
ACSC_INTR_PROGRAM_END	20	ACSPL+ program has finished.
ACSC_INTR_COMMAND	21	A line of ACSPL+ commands executed in a dynamic buffer.
ACSC_INTR_ACSPL_PROGRAM	22	ACSPL+ program has generated the interrupt by **INTERRUPT** command.
ACSC_INTR_MOTION_START	24	Physical motion has started.
ACSC_INTR_MOTION_PHASE_CHANGE	25	Motion profile changed the phase
ACSC_INTR_TRIGGER	26	AST.#TRIGGER bit went high
ACSC_INTR_NEWSEGM	27	AST.#NEWSEGM bit went high.

Name	Value	Description
ACSC_INTR_SYSTEM_ERROR	28	System error occurred.
ACSC_INTR_ETHERCAT_ERROR	29	EtherCAT error occurred.
ACSC_INTR_CYCLE	30	
ACSC_INTR_MESSAGE	31	
ACSC_INTR_COMM_CHANNEL_CLOSED	32	Communication channel has been closed.
ACSC_INTR_SOFTWARE_ESTOP	33	EStop button was clicked.

4.18 Callback Interrupt Masks

Syntax:

public enum AxisMasks: ulong

public enum BufferMasks: ulong

public enum InputMasks: uint

public enum OldAxisMasks: ulong

Table 4-21. Callback Interrupt Masks

Bit Name	Bit	Description	Interrupt
ACSC_NONE	0		
ACSC_ALL	-1		

Bit Name	Bit	Description	Interrupt
ACSC_MASK_AXIS_0 ... ACSC_MASK_AXIS_127	0 ... 127	Axis 0 ... Axis 127	ACSC_INTR_PEG, ACSC_INTR_MARK1, ACSC_INTR_MARK2, ACSC_INTR_PHYSICAL_MOTION_END, ACSC_INTR_LOGICAL_MOTION_END, ACSC_INTR_MOTION_FAILURE, ACSC_INTR_MOTOR_FAILURE, ACSC_INTR_MOTION_START, ACSC_INTR_MOTION_PHASE_CHANGE, ACSC_INTR_TRIGGER
ACSC_MASK_BUFFER_0 ... ACSC_MASK_BUFFER_63	0 ... 63	Buffer 0 ... Buffer 63	ACSC_INTR_PROGRAM_END, ACSC_INTR_COMMAND, ACSC_INTR_ACSPL_PROGRAM

Table 4-22. OldAxisMasks

Bit Name	Bit	Description
ACSC_NONE	0	
ACSC_ALL	-1	
ACSC_MASK_AXIS_X	0x00000001	
ACSC_MASK_AXIS_Y	0x00000002	
ACSC_MASK_AXIS_Z	0x00000004	
ACSC_MASK_AXIS_T	0x00000008	
ACSC_MASK_AXIS_A	0x00000010	
ACSC_MASK_AXIS_B	0x00000020	
ACSC_MASK_AXIS_C	0x00000040	
ACSC_MASK_AXIS_D	0x00000080	

4.19 Configuration Keys

Syntax:

public enum ConfigKey

Table 4-23. Configuration Keys

Key Name	Key	Description
ACSC_CONF _WORD1_KEY	1	Bit 6 defines HSSI route, bit 7 defines source for interrupt generation.
ACSC_CONF _INT_EDGE_KEY	3	Sets the interrupt edge to be positive or negative.
ACSC_CONF _ENCODER_KEY	4	Sets encoder type: A&B or analog.
ACSC_CONF_OUT_KEY	29	Sets the specified output pin to be one of the following: OUT0 PEG Brake
ACSC_CONF _MFLAGS9_KEY	204	Controls value of **MFLAGS.9**
ACSC_CONF_DIGITAL_SOURCE_KEY	205	Assigns use of OUT0 signal: general purpose output or PEG output.
ACSC_CONF_SP_OUT_PINS_KEY	206	Reads SP output pins.
ACSC_CONF_BRAKE_OUT_KEY	229	Controls brake method.

4.20 System Information Keys

Syntax:

public enum SystemInfoKey

Table 4-24. System Information Keys

Key Name	Key	Description
ACSC_SYS_MODEL_KEY	1	
ACSC_SYS_VERSION_KEY	2	

Key Name	Key	Description
ACSC_SYS_NBUFFERS_KEY	10	
ACSC_SYS_DBUF_INDEX_KEY	11	
ACSC_SYS_NAXES_KEY	13	
ACSC_SYS_NNODES_KEY	14	
ACSC_SYS_NDCCH_KEY	15	
ACSC_SYS_ECAT_KEY	16	

4.21 Representation Variables and Return Types Definitions

Syntax:

public enum AcsplVariableType

Table 4-25. ACSPL+ Variables Types

Name	Value	Description
ACSC_INT_TYPE	1	Integer Variable type
ACSC_REAL_TYPE	2	Real Variable type

Syntax:

public enum RepresentationTypes

Table 4-26. Representation Types

Name	Value	Description
ACSC_NONE	0	
ACSC_ALL	-1	
ACSC_DEC_REAL_TYPE	8	
ACSC_DEC_INT_TYPE	4	
ACSC_BIN_INT_TYPE	2	
ACSC_OCT_INT_TYPE	1	
ACSC_HEX_INT_TYPE	16	

4.22 Log Detalization and Presentation Definitions

Syntax:

public enum ACSC_LOG_DETALIZATION_LEVEL

Table 4-27. Log Detalization Definitions

Name	Value	Description
Minimum	0	
Medium	1	
Maximum	2	

Syntax:

public enum ACSC_LOG_DATA_PRESENTATION

Table 4-28. Log Presentation Definitions

Name	Value	Description
Compact	0	
Formatted	1	
Full	2	

4.23 FRF Enumerations

4.23.1 FRF_LOOP_TYPE

Description

Enumerates supported FRF loop types.

Syntax

```
typedef enum {
PositionVelocity=0,
Position=1,
Velocity=2,
Current=3,
Open=4
} FRF_LOOP_TYPE;
```

Arguments

PositionVelocity	Measures Plant and Sensitivity of the PositionVelocity loop along with measurement coherence. Measurement is executed in closed-loop mode.
Position	Measures Plant and Sensitivity of the Position loop along with measurement coherence. Measurement is executed in closed-loop mode.
Velocity	Measures Plant and Sensitivity of the Velocity loop along with measurement coherence. Measurement is executed in closed-loop mode.
Current	Measures Plant of the Current loop along with measurement coherence. Measurement is executed in closed-loop mode.
Open	- Measures Plant of the PositionVelocity loop along with measurement coherence. Measurement is executed in open-loop mode.

4.23.2 FRF_EXCITATION_TYPE

Description

Enumerates supported methods for distributing frequency points.

Syntax

```
typedef enum
{
 WhiteNoise = 0,
 ChirpPeriodic = 1,
 UserDefined = 2
} FRF_EXCITATION_TYPE;
```

Arguments

Linear	frequency points distributed linearly in the range defined by the startFreqHz and endFreqHz. Frequency resolution is determined by **durationSec** parameter
Logarithmic	– points are distributed according to **startFreqHz**, **endFreqHz**, **freqPerDec**, **highResolutionStart** and **highResolutionFreqPerDec**

4.23.3 FRF_WINDOW_TYPE

Description

The structure enumerates the supported windowing types.

Syntax

```
enum
{
 Hanning=0,
Rectangular=1,
Hamming=2
} FRF_WINDOW_TYPE;
```

Arguments

Hanning	Hanning filter of the measured signals. This filter may be used if excitation type is WhiteNoise $w(n) = \sin^2(n)$
Rectangular	No filtering. This filter should be used when excitation type is **ChirpPeriodic**.
Hamming	hamming filter of the output signals. This filter may be used if excitation type is WhiteNoise $w(n) = 0.54 + 0.46\cos(\frac{2\pi n}{N})$

4.23.4 FRF_FREQUENCY_DISTRIBUTION_TYPE

Description

The structure enumerates the frequency distribution types supported.

Syntax

```
enum
{
 Linear=0,
 Logarithmic=1
} FRF_FREQUENCY_DISTRIBUTION_TYPE;
```

Arguments

Linear	Frequency points distributed linearly in the range defined by the **startFreqHz** and **endFreqHz**. Frequency resolution is determined by **durationSec** parameter
Logarithmic	Points are distributed according to **startFreqHz**, **endFreqHz**, **freqPerDec**, **highResolutionStart** and **highResolutionFreqPerDec**

4.23.5 FRF_FREQUENCY_DISTRIBUTION_TYPE

Description

The structure enumerates the frequency distribution types supported.

Syntax

```
enum
{
 Linear=0,
 Logarithmic=1
} FRF_FREQUENCY_DISTRIBUTION_TYPE;
```

Arguments

Linear	Frequency points distributed linearly in the range defined by the **startFreqHz** and **endFreqHz**. Frequency resolution is determined by **durationSec** parameter
Logarithmic	Points are distributed according to **startFreqHz**, **endFreqHz**, **freqPerDec**, **highResolutionStart** and **highResolutionFreqPerDec**

4.23.6 FRF_CHIRP_TYPE

Description

Enumerates types of chirp input supported.

Syntax

```
typedef enum
{
 LogarithmicChirp,
LinearChirp
} FRF_CHIRP_TYPE;
```

Arguments

LogarithmicChirp	Chirp frequencies are distributed logarithmically in the time range. As a result, lower frequencies will have higher power per frequency.
LinearChirp	Chirp frequencies are distributed Linearly. As a result, lower all frequencies will have the same energy.

4.23.7 FRF_OVERLAP

Description

Enumerates the options for signal overlapping.

Syntax

```
typedef enum
{
 NoOverlap,
HalfSignal,
}FRF_OVERLAP;
```

Arguments

NoOverlap	Signals are not overlapped
HalfSignal	Signals are overlapped at half-length for better averaging in frequency domain

4.23.8 FRF_CROSS_COUPLING_TYPE

Description

Determines whether measurement of cross coupling is in closed or open loop configuration.

Syntax

```
typedef enum
{
Complete,
CompleteOpen
}FRF_CROSS_COUPLING_TYPE;
```

Arguments

Complete	Measures the cross coupling in a closed loop
CompleteOpen	Measures the cross coupling in an open loop

5. Events

SPiiPlus NET Library has events for the following types of interrupts:

The bit-mapped event parameter: **Param** is an interrupt mask that determines which axis/buffer/input a given interrupt was generated for. See Callback Interrupt Masks for a description of **Param** for each interrupt.

Table 5-1. Events and Interrupts

Event	Interrupt
ACSPLCOMMAND (long Param)	ACSC_INTR_COMMAND
ACSPLPROGRAM (BufferMasks Param)	ACSC_INTR_ACSPL_PROGRAM
COMMCHANNELCLOSED ()	ACSC_INTR_COMM_CHANNEL_CLOSED
EMERGENCY ()	ACSC_INTR_EMERGENCY
ETHERCATERROR ()	ACSC_INTR_ETHERCAT_ERROR
LOGICALMOTIONEND (AxisMasks Param)	ACSC_INTR_LOGICAL_MOTION_END
MOTIONFAILURE (AxisMasks Param)	ACSC_INTR_MOTION_FAILURE
MOTIONPHASECHANGE (AxisMasks Param)	ACSC_INTR_MOTION_PHASE_CHANGE
MOTIONSTART (AxisMasks Param).	ACSC_INTR_MOTION_START
MOTORFAILURE (AxisMasks Param)	ACSC_INTR_MOTOR_FAILURE
NEWSEGM ()	ACSC_INTR_NEWSEGM
PHYSICALMOTIONEND (AxisMasks Param)	ACSC_INTR_PHYSICAL_MOTION_END
PROGRAMEND (BufferMasks Param)	ACSC_INTR_PROGRAM_END
SOFTWAREESTOP ()	ACSC_INTR_SOFTWARE_ESTOP
SYSTEMERROR ()	ACSC_INTR_SYSTEM_ERROR
ACSPLPROGRAMEX(ulong Param)	ACSC_INTR_ACSPL_PROGRAM_EX
TRIGGER (AxisMasks Param)	ACSC_INTR_TRIGGER

6. Error Handling

The SPiiPlus NET Library throws **ACSException** when API function fails to complete the requested operation successfully. The exception object contains an error code and an Error Description that can be used to evaluate the error occurred.

> Error code, received from GetLastError() function (see Error Codes).

> A string with a description of the error.

For example:

```
Communication Initialization failure. Error – 132
```

6.1 Error Handling Example in C#

The following code illustrates error handling in C#:

```
private void OpenCommSerial()
{
 try {
     Ch.OpenCommSerial(1,115200);
 }
 catch (ACSException ex)
 {
   // handle Exception
 }
}
```

6.2 Error Codes

This section provides a breakdown of the Error codes associated with the .NET Library.

Any error code greater than 1000 is a controller error as defined in the *SPiiPlus Command & Variable Reference Guide*.

Table 6-1. NET Library Error Codes

Name	Error Code	Error Message	Remarks
ACSC_ERRORBASE	110		
ACSC_UNKNOWNERROR	100		

Name	Error Code	Error Message	Remarks
ACSC_ONLYSYNCHRONOUS	101	Asynchronous call is not supported.	Attempt was made to use a method asynchronously that is only support synchronously, for example, **Send**.
ACSC_ENOENTLOGFILE	102	No such file or directory.	This error is returned by the *OpenLogFile* method if a component of a path does not specify an existing directory.
ACSC_OLD_FW	103	The FW version does not support the current NET Library version.	This error is returned by one of the **OpenComm*** methods. Upgrade the FW of the controller.
ACSC_MEMORY_OVERFLOW	104	Controllers reply is too long.	A timeout has occurred due to a lack of response of the controller.
ACSC_EBADFLOGFILE	109	Internal library error: Invalid file handle.	
ACSC_RTOS_NOT_INITIALIZED	110		
ACSC_SHM_NOT_INITIALIZED	111		
ACSC_SHM_WRONG_TYPE	112		
ACSC_SHM_INVALID_ADDRESS	113		
ACSC_SHE_NOT_SUPPORTED	114		
ACSC_SHE_INITERROR	115		

Name	Error Code	Error Message	Remarks
ACSC_SHE_NOT_INITIALIZED	116		
ACSC_SHE_ARG_READ_ERROR	117		
ACSC_SHE_CLOSE_ERROR	118		
ACSC_EINVALLOGFILE	122	Internal library error: Cannot open Log file.	
ACSC_EMFILELOGFILE	124	Too many open files.	This error is returned by the *OpenLogFile* method if no more file handles available.
ACSC_ENOSPCLOGFILE	128	No space left on device.	This error is returned by the *WriteLogFile* method if no more space for writing is available on the device (for example when the disk is full).
ACSC_TIMEOUT	130	Timeout expired.	A time out occurred while waiting for a controller response. This error indicates that during specified timeout the controller did not respond or response was invalid.
ACSC_SIMULATOR_NOT_RUN	131		

Name	Error Code	Error Message	Remarks
ACSC_INITFAILURE	132	Communication initialization failure.	This error is returned by one of the **Open*** ** methods in the following cases: the specified communication parameters are invalid ,the corresponding physical connection is not established ,the controller does not respond for specified communication channel.
ACSC_SIMULATOR_RUN_ EXT	133	Internal library error: Creating communication object failure.	
ACSC_INVALIDHANDLE	134	Invalid communication handle.	Specified communication handle must be handle returned by one of the **Open*** ** methods.
ACSC_ALLCHANNELSBUSY	135	All channels are busy.	The maximum number of the concurrently opened communication channels is 10.
ACSC_SIMULATOR_NOT_ SET	136	Invalid name of Log file.	This error is returned by the *OpenLogFile* method if the specified log file name is invalid or more than 256 characters.
ACSC_RECEIVEDTOOLONG	137	Received message is too long (more than size of user buffer).	This error cannot be returned and is present for compatibility with previous versions of the library.

Name	Error Code	Error Message	Remarks
ACSC_INVALIDBUFSIZE	138	The program string is long.	This error is returned by one of the an *AppendBuffer* or *LoadBuffer* method if ACSPL+ program contains a string longer than 2032 bytes.
ACSC_ INVALIDPARAMETERS	139	Method parameters are invalid.	
ACSC_CLOSEDHISTORYBUF	140	History buffer is closed.	
ACSC_EMPTYNAMEVAR	141	Name of variable must be specified.	
ACSC_INPUTPAR	142	Error in index specification.	This error is returned by the *ReadVariable*, *ReadVariableAsScalar*, *ReadVariableAsVector*, *ReadVariableAsMatrix*, WriteVariable methods if the **From1**, **To1**, **From2**, **To2** arguements were specified incorrectly.
ACSC_RECEIVEDTOOSMALL	143	Controller reply contains less values than expected.	This error is returned by the *ReadVariable*, *ReadVariableAsScalar*, *ReadVariableAsVector*, *ReadVariableAsMatrix*, WriteVariable methods.
ACSC_ FUNCTIONNOTSUPPORTED	145	Function is not supported in current version	
ACSC_ INITHISTORYBUFFAILED	147	Internal error: Error of thehistory buffer initialization.	

Name	Error Code	Error Message	Remarks
ACSC_ CLOSEDMESSAGEBUF	150	Unsolicited messages buffer is closed.	
ACSC_SETCALLBACKERROR	151	Callback registration error.	This error is returned by the EnableEvent method for any of the communication channels. Only PCI Bus communication supports user callbacks.
ACSC_ CALLBACKALREADYSET	152	Callback method has been already installed.	This error is returned by the EnableEvent method if the application tries to enable another event for the same interrupt that was already used. Only one event can be enabled for each interrupt.
ACSC_CHECKSUMERROR	153	Checksum of the controllerresponse is incorrect.	
ACSC_ REPLIESSEQUENCEERROR	154	Internal library error: The controller replies sequence is invalid.	
ACSC_WAITFAILED	155	Internal library error.	
ACSC_ INITMESSAGEBUFFAILED	157	Internal library error: Error of the unsolicited messages buffer initialization.	
ACSC_ OPERATIONABORTED	158	Non-waiting call has been aborted.	

Name	Error Code	Error Message	Remarks
ACSC_ CANCELOPERATIONERROR	159	Error of the non-waiting call cancellation.	
ACSC_ COMMANDSQUEUEFULL	160	Internal error: Queue of transmitted commands is full.	
ACSC_SENDINGFAILED	162	The library cannot send to the specified communication channel.	Check physical connection with the controller (or settings) and try to reconnect.
ACSC_RECEIVINGFAILED	163	The library cannot receive from the specified communication channel.	Check physical connection with the controller (or settings) and try to reconnect.
ACSC_ CHAINSENDINGFAILED	164	Internal library error: Sending of the chain is failed.	
ACSC_DUPLICATED_IP	165	Specified IP address is duplicated.	
ACSC_APPLICATION_NOT_ FOUND	166	There is no Application with such Handle.	
ACSC_ARRAY_EXPECTED	167	Array name was expected.	
ACSC_INVALID_FILE_ FORMAT	168	File is not Data File.	Input file is not in ANSI format.
ACSC_APPSL_CRC	171	Application Saver Loader CRC Error	
ACSC_APPSL_HEADERCRC	172	Application Saver Loader Header CRC Error	

Name	Error Code	Error Message	Remarks
ACSC_APPSL_FILESIZE	173	Application Saver Loader File Size Error	
ACSC_APPSL_FILEOPEN	174	Application Saver Loader File Open Error	
ACSC_APPSL_ UNKNOWNFILE	175	Application Saver Loader Unknown File Error	
ACSC_APPSL_VERERROR	176	Application Saver Loader Format Version Error	
ACSC_APPSL_SECTION_SIZE	177	Application Saver Loader Section Size is Zero	
ACSC_TLSERROR	179	Internal library error: Thread local storage error.	
ACSC_INITDRIVERFAILED	180	PCI driver initialization error.	Returned by the *GetPCICards* method in the following cases: SpiiPlus PCI driver is not installed correctly. The version of the SpiiPlus PCI Bus driver is incorrect - in this case, it is necessary to reinstall the SpiiPlus PCI driver (WINDRIVER) and the library.
ACSC_CAN_INITFAILURE	181	CAN library not found or initialization failure	
ACSC_CLOSED_BY_ CONTROLLER	182	Communication closed by the controller	

Name	Error Code	Error Message	Remarks
ACSC_INVALIDPOINTER	185	Pointer to the buffer is invalid or Null pointer received instead of user allocated object	
ACSC_INVALIDPOINTER	189	Specified priority for the callback thread cannot be set	
ACSC_DIRECTDPRAMACCESS	190	Cannot access DPRAM directly through any channel but PCI and Direct.	Returned by DPRAM access methods, when attempting to call them with Serial or Ethernet channels.
ACSC_DDERROR	191		
ACSC_INVALID_DPRAM_ADDR	192	Invalid DPRAM address was specified	Returned by DPRAM access methods, when attempting to access illegal address
ACSC_OLD_SIMULATOR	193	This version of simulator does not support work with DPRAM.	Returned by DPRAM access methods, when attempting to access old version Simulator that does not support DPRAM.
ACSC_HW_PROBLEM	194		
ACSC_FILE_NOT_FOUND	195	File not found.	Returned by methods that work with host file system when a specified file name is not found. Check the path and filename.

Name	Error Code	Error Message	Remarks
	196	Not enough data	Returned by methods that analyze SPiiPlus application files when application file format is incorrect. Check application file and replace with a valid file.
ACSC_SERVEREXCEPTION	197	The application cannot establish communication with the SPiiPlus User Mode Driver	Returned by one of the **OpenComm** methods. Check the following: SPiiPlus User Mode Driver is loaded (that is, the User Mode Drive icon appears in the Task tray). SPiiPlus User Mode Driver shows an error message. In case of remote connection, access from a remote application is enabled.
ACSC_STOPPED_ RESPONDING	198	Not responding	The controller does not reply for more than 20 seconds. Returned by any method that exchanges data with the controller. Check the following: Controller is powered on (MPU LED is green) Controller connected properly to host Controller executes a time consuming command like compilation of a large program, save to flash, load to flash, etc.

Name	Error Code	Error Message	Remarks
ACSC_DLL_UMD_VERSION	199	The DLL and the UMD versions are not compatible.	Returned by one of the OpenComm methods. Verify that the files ACSCL_x86.DLL / ACSCL_x64.DLL and ACSCSRV.EXE are of the same version.
ACSC_FRF_INPUT_START_FREQUENCY_OUT_OF_RANGE	200		
ACSC_FRF_INPUT_END_FREQUENCY_OUT_OF_RANGE	201		
ACSC_FRF_INPUT_START_FREQUENCY_IS_HIGHER_THAN_END_FREQUENCY	202		
ACSC_FRF_INPUT_FREQPERDEC_OUT_OF_RANGE	203		
ACSC_FRF_INPUT_HR_FREQPERDEC_OUT_OF_RANGE	204		
ACSC_FRF_INPUT_FREQUENCY_RESOLUTION_LINEAR_OUT_OF_RANGE	205		
ACSC_FRF_INPUT_AMPLITUDE_OUT_OF_RANGE	206		
ACSC_FRF_INPUT_AXIS_OUT_OF_RANGE	207		
ACSC_FRF_INPUT_NUMBER_OF_REPETITIONS_OUT_OF_RANGE	208		

Name	Error Code	Error Message	Remarks
ACSC_FRF_INPUT_DURATION_OUT_OF_RANGE	209		
ACSC_FRF_INPUT_ENUM_VALUE_OUT_OF_RANGE	210		
ACSC_FRF_MEMORY_ALLOCATION_FAILED_AT_HOST	211		
ACSC_FRF_DATA_READ_FROM_CONTROLLER_INCONSISTENT	212		
ACSC_FRF_DSP_DOESNT_HAVE_REQUIRED_PARAMETERS	213		
ACSC_FRF_FAILED_TO_COMMUNICATE_WITH_CONTROLLER	214		
ACSC_FRF_FAILED_TO_READ_SERVO_PARAMETERS	215		
ACSC_FRF_DUMMY_AXIS_NOT_SUPPORTED	216		
ACSC_FRF_MOTOR_SHOULD_BE_SET_TO_CLOSED_LOOP	217		
ACSC_FRF_MOTOR_SHOULD_BE_ENABLED	218		
ACSC_FRF_MOTOR_SHOULD_COMMUTATED	219		
ACSC_FRF_SPDC_IS_ALREADY_IN_PROGRESS	220		

Name	Error Code	Error Message	Remarks
ACSC_FRF_ABORTED_BY_USER	221		
ACSC_FRF_MOTOR_DISABLED_DURING_MEASUREMENT	222		
ACSC_FRF_DISABLE_OR_FAULT_OCCURED_DURING_MEASUREMENT	223		
ACSC_FRF_FAULT_OCCURED_DURING_MEASUREMENT	224		
ACSC_FRF_ARRAY_SIZES_INCOMATIBLE	225		
ACSC_FRF_NUMBER_OF_POINTS_SHOULD_BE_POSITIVE	226		
ACSC_FRF_MEMORY_ALLOCATION_FAILED_AT_CONTROLLER	227		
ACSC_FRF_EXCITATION_DURATION_IS_TOO_LONG	228		
ACSC_FRF_USER_DEFINED_EXCITATION_SIGNAL_REQUIRED_BUT_NOT_DEFINED	229		
ACSC_FRF_USER_DEFINED_EXCITATION_SIGNAL_OUT_OF_BOUNDARIES	230		
ACSC_FRF_FRD_LENGTH_TOO_SHORT	231		

Name	Error Code	Error Message	Remarks
ACSC_FRF_FRD_FREQUENCIES_SHOULD_BE_CONTINUOUSLY_INCREASING	232		
ACSC_JITTER_ANALYSIS_JITTER_ARRAY_TOO_SHORT	233		
ACSC_JITTER_ANALYSIS_SAMPLING_FREQUENCY_NOT_VALID	234		
ACSC_JITTER_ANALYSIS_WINDOW_TYPE_NOT_SUPPORTED	235		
ACSC_JITTER_ANALYSIS_FREQUENCY_RANGE_NOT_VALID	236		
ACSC_JITTER_ANALYSIS_FREQUENCY_RESOLUTION_NOT_VALID	237		
ACSC_LICENSE_COMMON_PROBLEM	238		
ACSC_LICENSE_DONGLE_NOT_FOUND	239		
ACSC_LICENSE_ENTRY_NOT_FOUND	240		
ACSC_LICENSE_INVALID_HANDLE	241		
ACSC_LICENSE_NO_DATA_AVAILABLE	242		
ACSC_LICENSE_INVALID_PN	243		

Name	Error Code	Error Message	Remarks
ACSC_SC_INCORRECT_PROC_ALLOC	244		
ACSC_SC_MISSING_DRIVERS	245		
ACSC_SC_INCORRECT_MEMORY	246		
ACSC_SC_RTOS_SERVICE	247		
ACSC_SC_REBOOT	248		
ACSC_SC_DONGLE_VERSION	249		
ACSC_LICENSE_NONLICENSED_FEATURE_MATLAB	250		
ACSC_LICENSE_NONLICENSED_FEATURE_FRF	251		
ACSC_LICENSE_NONLICENSED_FEATURE_COMMON	252		
ACSC_FRF_GENERAL_ERROR	253		
ACSC_HW_ERRORBASE	500		
ACSC_HW_NO_INT	502		
ACSC_HW_INT_PERIOD	504		
ACSC_HW_NO_INT_NOTIF	506		
ACSC_HW_SPiiFAILURE	508		
ACSC_CANDEVICE_CUSTOM1	1		

Name	Error Code	Error Message	Remarks
ACSC_CANDEVICE_CUSTOM2	2		
ACSC_CANDEVICE_NI	11		
	601	Error in array and/or index definition	
	602	Communication channel is already open	
	603	Argument Value has incorrect type	
	604	Argument Value is not initialized	
	605	Variable name must be specified	
	606	Wrong call type was specified	
	607	Wrong return type	

Smarter Motion

5 HaTnufa St.
Yokne'am illit 2066717
Israel
Tel: (+972) (4) 654 6440 Fax: (+972) (4) 654 6443

ACS MOTION CONTROL