CHAPTER

11

Auto Loader and MC Loader ActiveX

Project Autoloader

Trio Project Autoloader is a stand alone *Motion Coordinator* program to load projects created using *Motion* Perfect 2 onto a Trio *Motion Coordinator*

The program is intended for easy loading of projects onto controllers without the need to run *Motion* Perfect and so allows OEM manufacturers to update customers equipment easily.

Operation of the program is controlled using a script file which gives a series of commands to be processed, in order, by the program.

Using the Autoloader

General

The autoloader is primerally intended to be used to update controllers already installed in equipment to allow OEM manufacturers to update customers equipment easily. It can be used from a hard disk or CD-ROM.

Script File

The autoloader program uses a script file AutoLoader.tas as a source of commands. These commands are executed in order until all commands have been processed or an error has occurred.

If any command fails the exececution terminates without completing the scripted command sequence.

Project

The project to be loaded using LOADPROJECT is in the form of a normal *Motion* Perfect 2 project. This consists of a directory containing a project definition file and Trio BASIC program files. The directory must have the same name as the project definition file less the extension.

i.e. project definition file TestProj.prj, directory TestProj

The project directory must be in the LoaderFiles directory.

Timeout

If there are large programs in the project the command timeout may need to be increased from its default value of 10 seconds otherwise the project load may fail due to the long time it takes to select a new program on the controller. The TIMEOUT command should appear in the script file before any LOADPROJECT command.

Tables

Any tables to be loaded must be in the form of *.lst files produced by Motion Perfect 2.

Normally these table files will be in the LoaderFiles directory.

Extra Programs

Programs which need to be loaded using LOADPROGRAM because they are not in the project being loaded (or if no project is being loaded)

Normally these program files will be in the LoaderFiles directory.

Files

The autoloader is designed to work with the following file structure (fixed names are shown in bold type).

Base Directory	**AutoLoader.exe**		
	LoaderFiles	**AutoLoader.tas**	
		Project	Project.**prj**
			Prog1.**bas**
			Prog2.**bas**
		Table1.**lst**	
		ExtProg1.**bas**	

Where:

Base Directory is normally the root directory, but can be any directory.

Project is the Motion Perfect 2 project directory for the project to be loaded using the LOADPROJECT command, Project**.prj** being the project file and Proj?**.bas** are the program files in the project.

Table?**.lst** are the table files to be loaded using the LOADTABLE command.

ExtProg?**.bas** are the extra programs to be loaded using the LOADPROGRAM command.

Any or all of the objects in the LoaderFiles directory can be located elsewhere as long as the file (or directory) name is specified using a full path. The script file can be specified as a single argument to the AutoLoader program.

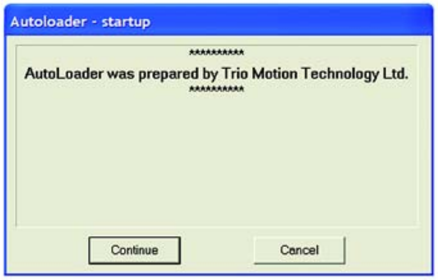
Running the program

The program can be started in the same way as any other Windows program, in which case the LoaderFiles directory must be in the same directory as the AutoLoader executable file.

It can also be started from the command line with an optional argument which specifies the script file to process. e.g.

```
AutoLoader E:\MXUpdate\20051203\UpDate1.tas
```

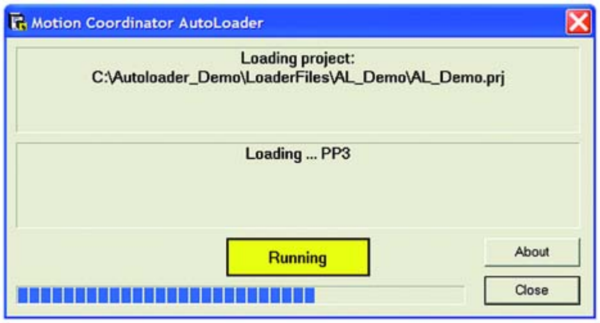
Start Dialog



****
AutoLoader was prepared by Trio Motion Technology Ltd.
****

The start dialog displays a message specified in the script and has **continue** and **cancel** buttons so that the user can exit from the program without running the script.

Main Window



Loading project:
C:\Autoloader_Demo\LoaderFiles\AL_Demo\AL_Demo.prj

Loading ... PP3

The program main window consists of two message windows; one to display the current command and the other to display the name of the program or file currently being loaded. There is a button to show the current status (Starting, running, pass or fail) and a progress bar to show the progress during file and table loading.

The **close** button closes the dialog. If it is pressed while a script is being processed then script processing will be terminated at the end of the current operation.

Script Commands

The following commands are available for use in script files

AUTORUN
CHECKPROJECT
CHECKTYPE
CHECKUNLOCKED
CHECKVERSION
COMMLINK (alternative COMMPORT)
COMPILEALL
COMPILEPROGRAM
DELETEALL (alternative NEWALL)
EPROM
FASTLOADPROJECT
HALTPROGRAMS
LOADPROGRAM
LOADPROJECT
LOADTABLE
SETPROJECT
SETRUNFROMEPROM
STARTUPMESSAGE
TIMEOUT
DELTABLE

All commands return a result of **OK** or **Fail**. An **OK** result allows script execution to continue, a **Fail** result will make script execution terminate at that point.

AUTORUN

Purpose: To run the programs on the controller which are set to run automatically at power-on.

Syntax: **`AUTORUN`**

CHECKPROJECT

Purpose: To check the programs on a controller against a project on disk.

Syntax: **`CHECKPROJECT [<ProjectName>]`**

Where <ProjectName> is the optional path of the project directory. If the project directory is in the same directory as the ALoader.exe executable then it is just the name of the of the project directory. If no <ProjectName> is specified then the current project, set by a previous SETPROJECT or LOADPROJECT command, is used. This operation is automatically performed by a LOADPROJECT operation.

Examples:

```
CHECKPROJECT
CHECKPROJECT TestProj
```

CHECKTYPE

Purpose: To check the controller type.

Syntax: **`CHECKTYPE <Controller List>`**

Where <Controller List> is a comma separated list of one or more valid controller ID numbers.

i.e. 206,216

Examples:

```
CHECKTYPE 206
CHECKTYPE 202,216,206
```

Controller ID Numbers

Each type of controller returns a different ID number in response to the TrioBASIC command ?CONTROL[0] . The table below gives the ID number for current controllers.

Controller	CONTROL
MC302X	293
Euro205x	255
Euro209	259
MC206X	207
PCI208	208
MC224	224

CHECKUNLOCKED

Purpose: To check that the controller is not locked.

Syntax: `CHECKUNLOCKED`

CHECKVERSION

Purpose: To check the version of the controller system code.

Syntax:

```
CHECKVERSION <Operator><Version>
CHECKVERSION <LowVersion>-<HighVersion>
```

Examples:

```
CHECVERSION >1.49
CHECKVERSION >= 1.51
CHECKVERSION 1.42-1.50
```

Comment

Purpose: To allow the user to put descriptive comments into a script.

Syntax: `' <Text>`

Where <Text> is any text.

Examples:
```
' This is a comment line
```

COMMLINK (alternative COMMPORT)

Purpose: To set the communications port and parameters.

Syntax: **Serial**

For a serial port this string is similar to COM1:9600,7,e,2 to specify the port, speed, number of data bits, parity and number of stop bits. 9600,7,e,2 are the default parameters for a controller.

USB

For a USB connection the string is USB:0 as only a single USB connection (0) is supported.

Ethernet

For an ethernet connection the string is similar to Ethernet:192.168.0.123:23 which specifies an ethernet connection to IP address 192.168.0.123 on port 23. The final ':' and the port number can be omitted, in which the port number defaults to 23.

PCI

For a PCI connection the string is similar to PCI:0 which specifies a connection to PCI card 0.

Examples:
```
COMMLINK COM2:9600,7,e,2
COMMLINK USB:0
COMMLINK Ethernet:192.168.0.111
COMMLINK PCI:1
```

COMPILEALL

Purpose: To compile all the programs on the controller.

Syntax: **`COMPILEALL`**

COMPILEPROGRAM

Purpose: To compile a program on the controller.

Syntax: **`COMPILEPROGRAM <Program>`**
Where <Program> is the program name.

Examples: **`COMPILEPROGRAM Prog`**

Note: The **`LOADPROGRAM`** command automatically compiles programs after thay are loaded so under normal circumstances there is no need to use this command.

DELETEALL (alternative NEWALL)

Purpose: To delete all programs on the controller.

Syntax: **`DELETEALL`**

EPROM

Purpose: To store the project currently in controller RAM into EPROM

Syntax: **`EPROM`**

FASTLOADPROJECT

Purpose: To load a project from disk onto the controller.

Description: FASTLOADPROJECT is a faster alternative to LOADPROJECT. It is only compatible with system software version 1.63 or later for '2' series Motion Coordinators, and version 1.9013 or later for '3' series Motion Coordinators.

FASTLOADPROJECT must be used if a project contains encrypted programs.

Syntax: `FASTLOADPROJECT [<ProjectName>]`

Where <ProjectName> is the optional path of the project directory. If the project directory is in the same directory as the ALoader.exe executable then it is just the name of the of the project directory. If no <ProjectName> is specified then the current project, set by a previous SETPROJECT command, is used.

Examples:

```
FASTLOADPROJECT
FASTLOADPROJECT TestProj
```

Note: If FASTLOADPROJECT fails and the project only contains TrioBASIC source files then using LOADPROJECT may work.

HALTPROGRAMS

Purpose: To halt all programs on the controller.

Syntax: `HALTPROGRAMS`

This operation is automatically performed as part of LOADPROJECT, LOADPROGRAM and DELTABLE commands.

LOADPROJECT

Purpose: To load a project from disk onto the controller.

Syntax: `LOADPROJECT <ProjectName>`

Where <ProjectName> is the optional path of the project directory. If the project directory is in the same directory as the ALoader.exe executable then it is just the name of the of the project directory. If no <ProjectName> is specified then the current project, set by a previous SETPROJECT command, is used.

Examples:

```
LOADPROJECT
```

LOADPROJECT TestProj

Note: LOADPROGRAM will only load TrioBASIC soruce files.

LOADPROGRAM

Purpose: To load a program not in a project onto the controller.

Syntax: **LOADPROGRAM <ProgramFile>**

Where <ProgramFile> is the path of the program file. If the program file is in the same directory as the ALoader.exe executable then this is just the file name of the program file.

Examples: **LOADPROGRAM TestProg.bas**

Note: LOADPROGRAM will only load TrioBASIC soruce files.

LOADTABLE

Purpose: To load a table onto the controller.

Syntax: **LOADTABLE <TableFile>**
Where <TableFile> is the path of the table file. If the table file is in the Loader-Files directory then this is just the file name of the table file.

This command should always be used after the LOADPROJECT command.

Examples: **LOADTABLE Tbl.lst**

SETPROJECT

Purpose: To set the current project for following commands.

Syntax: **SETPROJECT <ProjectName>**

Where <ProjectName> is the path of the project directory. If the project directory is in the same directory as the ALoader.exe executable then it is just the name of the of the project directory.

Examples:

```
SETPROJECT TestProj
```

SETRUNFROMEPROM

Purpose: To set the controller to use the programs stored in its EPROM. (It actually copies the programs from EPROM into RAM at startup).

Syntax:

```
SETRUNFROMEPROM <State>
```

Where <State> is **1** for copy from EPROM and **0** is use programs currently in RAM.

A single @ character can be used to specify state in the project file.

Examples:

```
SETRUNFROMEPROM 1
SETRUNFROMEPROM @
```

Note: This command only applies to controllers which have battery backed RAM (controllers with no battery backed RAM will always copy programs from EPROM).

Startup Message

Purpose: To allow the user to display a custom message in the startup dialog.

Multiple lines can be used to specify the message, they are displayed in the order that they appear in the script file. The message can be specified anywhere in the script file and the lines need not be together in the file.

Syntax:

```
# <Text>
```

Where <Text> is any text.

Examples:

```
# ***
# This autoloader was set up by ABCD Inc. to change Valve
Machine to left-hand thread
# ***
```

TIMEOUT

Purpose: To set the command timeout.

Syntax: **`TIMEOUT time`**

Where time is the timeout value in seconds (default is 10).

Examples: **`TIMEOUT 30`**

Note: It will normally only be necessary to increase the timeout above 10 if there are large programs in the target controller or you are loading large programs onto it.

Script File

The autoloader program uses a script file AutoLoader.tas as a source of commands. These commands are executed in order until all commands have been processed or an error has occurred.

If any command fails the exececution terminates without completing the scripted command sequence.

Sample Script

```
  ' Test Script
  ' **************
  ' Startup Message
  # ***
  # This autoloader was set up by TRIO to load a test project
onto a controller of fixed type.
  # ***
  COMMLINK COM1:9600,7,e,2
  CHECKTYPE 206
  CHECKVERSION > 1.45
  CHECKUNLOCKED
  LOADPROJECT LoaderTest
  LOADTABLE tbl_1.lst
  OPEN
  CHECKPROJECT LoaderTest
  LOADPROGRAM flashop.bas
  LOADPROGRAM clrtable.bas
  LOADPROGRAM settable.bas
  EPROM
  SETRUNFROMEPROM @
```

For this script to work correctly the LoaderFiles directory must contain a project directory LoaderTest, a table file tbl_1.lst and three program files: flashop.bas, clrtable.bas and settable.bas.

MC Loader

Trio MC Loader is a Windows ActiveX control which can load projects (produced with Motion Perfect) and programs onto a Trio *Motion Coordinator*. Communication with the *Motion Coordinator* can be via Serial link, USB, Ethernet or PCI depending on the *Motion Coordinator*.

Requirements

- PC with one or more of USB interface, Ethernet network interface, or PCI based *Motion Coordinator.*
- Windows 98, ME, 2000 or XP (Windows 2000 or XP only for PCI connection)
- TrioUSB driver - for USB connection
- Trio PCI driver - for PCI connection (Windows 2000 and XP systems only)
- Knowledge of the Trio *Motion Coordinator* to which the TrioPC ActiveX controls will connect.
- Knowledge of the Trio BASIC programming language.

Installation of the MC Loader Component

Launch the program "Install_TrioMCLoader" and follow the on-screen instructions. The TrioUSB driver and TrioPC ocx will be installed and registered to your Windows environment. The Trio MC Loader driver will also be installed on systems running Windows 2000 or Windows XP. A Windows Help file is included as an alternative to the printed pages in this manual.

Using the Component

The MC Loader component must be added to the project within your programming environment. Here is an example using Visual Basic, however the exact sequence will depend on the software package used.

From the Menu select Project then Components... (or use shortcut ctrl+T).

When the Components dialogue box has opened, scroll down until you find "Trio MC Loader Control Module" then click in the block next to Trio MC Loader. (A tick will appear)

Now click OK and the component should appear in the control panel on the left side of the screen. It is identified as TrioMCLoader Control.

Once you have added the Trio MC Loader component to your form, you are ready to build the project and include the Trio MC Loader methods in your programs.

Properties

The control has the following properties:

`CommLink`
`ControllerType`
`ControllerSystemVersion`
`DecryptionKey`
`Locked`
`ProjectFile`
`RunFromEPROM`
`Timeout`

Events

The control does not generate any events.

CommLink

Type: BSTR (string)

Access: Read / write

Description: This property is used to get or set the configuration of the communications link. The format of the string depends on the type of communications link being used.

Serial

For a serial port this string is similar to COM1:9600,7,e,2 to specify the port, speed, number of data bits, parity and number of stop bits. 9600,7,e,2 are the default parameters for most controllers.

USB

For a USB connection the string is USB:0 as only a single USB connection (0) is supported.

Ethernet

For an ethernet connection the string is similar to Ethernet:192.168.0.123:23 which specifies an ethernet connection to IP address 192.168.0.123 on port 23. The final ':' and the port number can be omitted, in which the port number defaults to 23.

PCI

For a PCI connection the string is similar to PCI:0 which specifies a connection to PCI card 0.

Examples: **Visual BASIC:**

```
axLoader.CommLink = "Ethernet:192.168.22.11"
```

Visual C#:

```
axLoader.CommLink = "Ethernet:192.168.22.11";
```

ControllerType

Type: unsigned long

Access: Read

Description: This is a read-only property which returns the Controller Type code.

Examples: **Visual BASIC:**

```
Dim ConType As Long
ConType = axLoader.ControllerType
```

Visual C#:

```
ulong ulConType;
ulConType = axLoader.ControllerType;
```

ControllerSystemVersion

Type: double

Access: Read

Description: This is a read-only property which returns the controller system software version number.

Examples: **Visual BASIC:**

```
Dim Version As Double
Version = axLoader.ControllerSystemVersion
```

Visual C#:

```
double dVersion;
dVersion = axLoader.ControllerSystemVersion;
```

DecryptionKey

Type: BSTR (string)

Access: Read / write

Description: The DecryptionKey property sets/gets the decryption key for a subsequent fast mode LoadProject operations. The decryption key is only used when a project containing one or more encrypted programs is loaded onto a controller using fast LoadProject.

Examples: **Visual BASIC:**

```
axLoader.DecryptionKey = "hjiHU87OOo"
```

Visual C#:

```
axLoader.DecryptionKey = "hjiHU87OOo";
```

Note: Decryption keys are a derived from the key string used to encrypt the program(s) and the security code of the target controller. Decryption keys can be generated using the Project Encryptor tool distributed with Motion Perfect.

Locked

Type: VARIANT_BOOL

Access: Read

Description: This is a read-only property which returns the locked state of the controller (true for locked, false for unlocked).

Examples: **Visual BASIC:**

```
Dim IsLocked As Boolean
IsLocked = axLoader.Locked
```

Visual C#:

```
bool bLocked;
bLocked = axLoader.Locked;
```

Open

Type: bool

Access: Read / write

Description: The Open property sets/gets the state of the communications port used to communicate with the controller.

Examples: **Visual BASIC:**

```
If Not axLoader.Open Then
  axLoader.Open = False
End If
```

Visual C#:

```
if (!axLoader.Open)
  axLoader.Open = false;
```

Note: Any method or property which needs to communicate with the controller will automatically open a communications port if the parameters have been set. The communications port is not closed on completion of a command so the primaty use of this property is to close the communications link rather than to open it.

ProjectFile

Type: BSTR (string)

Access: Read / write

Description: This property is used to get or set the current project file. The full path to the project file should be used when setting this property.

Examples: **Visual BASIC:**

```
If axLoader.ProjectFile.Length = 0 then
    axLoader.ProjectFile = "C:\Projects\PPX\PPX.prj"
End If
```

Visual C#:

```
if (axLoader.ProjectFile.Length == 0)
    axLoader.ProjectFile = "C:\\Projects\\PPX\\PPX.prj";
```

RunFromEPROM

Type: VARIANT_BOOL

Access: Read / write

Description: This property is used to control how the controller starts up. When set to false it uses programs stored in its RAM memory. When set to true the controller uses programs stored in its EPROM memory (overwriting the programs in RAM).

Examples: **Visual BASIC:**

```
If not axLoader.RunFromEPROM then
    axLoader.RunFromEPROM = True
End If
```

Visual C#:

```
if (!axLoader.RunFromEPROM)
    axLoader.RunFromEPROM = true;
```

Timeout

Type:	unsigned long
Access:	Read / write
Description:	This property is used to set the command timeout for communications with the controller. The default value is 10 (seconds) but may need to be increased if you are using large programs or have a large project.

Examples: **Visual BASIC:**

```
If axLoader.Timeout < 20 Then
    axLoader.Timeout = 25
End If
```

Visual C#:

```
if (axLoader.Timeout < 20)
    axLoader.Timeout = 25;
```

Methods

The control has the following methods:

AutoRun
CheckProject
CompileAll
CompileProgram
DeleteAll
DeleteTable
GetLastError
GetLastErrorString
HaltPrograms
LoadProgram
LoadProject
LoadTable
Lock
Unlock

AutoRun

Parameters: none

Return Type: VARIANT_BOOL

Description: This method is used to run any programs on the controller which are set to auto-run on startup.

The return value is true if the method call succeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: **Visual BASIC:**

```
If Not axLoader.AutoRun Then
 DisplayError(axLoader.GetLastError,
axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.AutoRun())
DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

CheckProject

Parameters: none

Return Type: VARIANT_BOOL

Description: This method is used to check the programs on the controller against the project previously set using the ProjectFile.

The return value is true if the method call succeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: **Visual BASIC:**

```
If Not axLoader.CheckProject Then
 DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.CheckProject())
DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

CompileAll

Parameters: none

Return Type: VARIANT_BOOL

Description: This method is used to compile all programs on the controller.

The return value is true if the method call succeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: **Visual BASIC:**

```
If Not axLoader.CompileAll Then
 DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.CompileAll())
DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

CompileProgram

Parameters: BSTR (string): ProgramName

Return Type: VARIANT_BOOL

Description: This method is used to compile a single program on the controller.

The return value is true if the method call succeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: **Visual BASIC:**

```
If Not axLoader.CompileProgram("PROG") Then
 DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.CompileProgram("PROG"))
DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

DeleteAll

Parameters: none

Return Type: VARIANT_BOOL

Description: This method is used to delete the all the programs on the controller.

The return value is true if the method call succeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: **Visual BASIC:**

```
If Not axLoader.DeleteAll Then
 DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.DeleteAll())
DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

DeleteTable

Parameters: none

Return Type: VARIANT_BOOL

Description: This method is used to delete the table on the controller. It only works on controllers which do not have dedicated table memory.

The return value is true if the method call succeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.Examples:

Visual BASIC:

```
If Not axLoader.DeleteTable Then
 DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.DeleteTable())
DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

GetLastError

Parameters: none

Return Type: unsigned long

Description: This method is used to retrieve the error code after a method call has failed (returned false). The returned error code is only valid for the previous method call. The following error codes can be returned:

Code	Error Description
0	No error
1	File does not exist
2	Error opening file
3	Invalid IP address
4	Invalid IP port
5	Invalid integer
6	Invalid communications port
7	Invalid communications parameters
8	Communications error
9	Invalid controller system version
10	Invalid controller type
11	Controller type not found
12	Invalid range
13	Failed version check
14	Controller locked
15	Failed to set project
16	Invalid command
17	Directory does not exist
18	No file specified
19	Program not in project
20	Program not on controller
21	CRC mismatch
22	Invalid directory
23	Failed to create directory
24	Invalid program file name
25	Error writing to file
26	Error reading CRC
27	Error calculating CRC

Code	Error Description
28	File not in project
29	Invalid program name
30	Failed to halt programs
31	Error reading directory
32	Program faild to compile
33	Failed to set communications parameters
34	Failed to get communications parameters
35	Transmit failure
36	Invalid connection type
37	Internal pointer error
38	Error sending string
39	Error sending command
40	Failed to select program

Further error information can be obtained by calling the GetLastErrorString method.

Examples: **Visual BASIC:**

```
If Not axLoader.CompileAll Then
 DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.CompileAll())
DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

GetLastErrorString

Parameters: none

Return Type: BSTR (string)

Description: This method is used to retrieve additional information from the controller. The string contains extra information which can be used in conjunction with the error code returned by the GetLastError method.

Examples: **Visual BASIC:**

```
If Not axLoader.CompileAll Then
```

```
 DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.CompileAll())
DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

HaltPrograms

Parameters: none

Return Type: VARIANT_BOOL

Description: This method is used to halt all programs currently running on the controller.

The return value is true if the method call succeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: **Visual BASIC:**

```
If Not axLoader.HaltPrograms Then
 DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.HaltPrograms())
DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

LoadProgram

Parameters: BSTR (string): ProgramFileName
VARIANT_BOOL: Compile

Return Type: VARIANT_BOOL

Description: This method is used to load a single program onto the controller. It is generally good practice to compile after loading the program.

The return value is true if the method call succeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: **Visual BASIC:**

```
If Not axLoader.LoadProgram("C:\Programs\Prog.bas", True) Then
 DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.LoadProgram("C:\\Programs\\Prog.bas", true))
DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

LoadProject

Parameters: VARIANT_BOOL: FastLoad

Return Type: VARIANT_BOOL

Description: This method is used to load the project previously set using the ProjectFile property onto the controller. If FastLoad is true, the loader will use the fast loading algorithm. Fast loading is not available some controllers and is only available in more recent versions of system software. All controllers will perform a normal (slow) load. Fast load must be used if the project contains one or more encrypted programs.

The return value is true if the method call succeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: **Visual BASIC:**

```
If Not axLoader.LoadProject(False) Then
 DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.LoadProject(false))
DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

LoadTable

Parameters: BSTR (string): TableFileName

Return Type: VARIANT_BOOL

Description: This method is used to load data into the table on the controller from a table list file (usually saved by Motion Perfect).

The return value is true if the method call succeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: **Visual BASIC:**

```
If Not axLoader.LoadTable("C:\Tables\ThisTable.lst") Then
 DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.LoadTable("C:\\Tables\\ThisTable.lst"))
DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

Lock

Parameters: unsigned long: Lock Code

Return Type: VARIANT_BOOL

Description: This method is used to lock the controller so that programs cannot be edited. The lock code used here must also be used if the controller is unlocked using the Unlock method.

The return value is true if the method call succeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: **Visual BASIC:**

```
If Not axLoader.Lock(1234) Then
 DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.Lock(1234))
DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```

Unock

Parameters: unsigned long: LockCode

Return Type: VARIANT_BOOL

Description: This method is used to unlock a locked controller so that programs can be edited. The lock code used here must be the same as the code used to lock the controller.

The return value is true if the method call succeded and false if it failed. Further error information can be obtained by calling the GetLastError and GetLastErrorString methods.

Examples: **Visual BASIC:**

```
If Not axLoader.Unlock(1234) Then
 DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString)
End If
```

Visual C#:

```
if (!axLoader.Unlock(1234))
DisplayError(axLoader.GetLastError,axLoader.GetLastErrorString);
```