Automate API Programmers Guide

2012

Contents

Revision History........3
EULA........4
1 Introduction........6
2 The architecture of the Automate........7
3 Installing the USB-driver........9
4 Using the DLL........10
5 CommunicationCls........11
5.1 Summary of the properties and methods........11
5.2 Properties........12
5.3 Events........14
5.4 Methods........14
6 ControlCls........16
6.1 Summary of the properties and methods........16
6.2 Properties........18
6.3 Events........20
6.4 Methods........21
7 InstrumentCls........46
7.1 Summary of the properties and methods........46
8 Registers........48
8.1 Introduction........48
8.2 Present register........48
8.3 Master Status register........49
8.4 Error registers........50
8.4.1 The Master Error Register........50
8.4.2 The Slave Error Registers........50
8.5 Tracing the Error........51

Revision History

Revision	Date	Changes
1.0	02.01.2010	First release update
1.01	dd.mm.2010	Update
1.07	02.12.2010	Update
1.30.02	14.11.2011	Update
1.3.2	22.2.2012	Update

EULA

This end-user license agreement is for the Biohit Oyj InstrumetLib.dll, intended to control the operation of the Biohit Liquid handling automate - products.

PLEASE READ THE TERMS AND CONDITIONS OF THIS LICENSE AGREEMENT CAREFULLY BEFORE INSTALLING OR USING THE SOFTWARE PRODUCT.

End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Biohit Oyj ("BIOHIT"), for the software product(s) identified above which may include associated software components, media, printed materials, and "online" or electronic documentation ("SOFTWARE PRODUCT"). By installing, copying, or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA. This license agreement represents the entire agreement concerning the program between you and Biohit Oyj, (referred to as "licenser"), and it supersedes any prior proposal, representation, or understanding between the parties. If you do not agree to the terms of this EULA, do not install or use the SOFTWARE PRODUCT.

The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.

1. GRANT OF LICENSE.
The SOFTWARE PRODUCT is licensed as follows:

(a) Installation and Use.
BIOHIT grants you the right to install and use copies of the SOFTWARE PRODUCT on your computer running a validly licensed copy of the operating system for which the SOFTWARE PRODUCT was designed [e.g., Windows XP/Vista/7].

(b) Backup Copies.
You may also make copies of the SOFTWARE PRODUCT as may be necessary for backup and archival purposes.

2. DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS.

(a) Maintenance of Copyright Notices.
You must not remove or alter any copyright notices on any and all copies of the SOFTWARE PRODUCT.

(b) Distribution.
You may not distribute registered copies of the SOFTWARE PRODUCT to third parties.

(c) Prohibition on Reverse Engineering, Decompilation, and Disassembly.
You may not reverse engineer, decompile, or disassemble the SOFTWARE PRODUCT, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation.

(d) Rental.
You may not rent, lease, or lend the SOFTWARE PRODUCT.

(e) Support Services.
BIOHIT may provide you with support services related to the SOFTWARE PRODUCT ("Support Services"). Any supplemental software code provided to you as part of the Support Services shall be considered part of the SOFTWARE PRODUCT and subject to the terms and conditions of this EULA.

(f) Compliance with Applicable Laws.
You must comply with all applicable laws regarding use of the SOFTWARE PRODUCT.

3. TERMINATION
Without prejudice to any other rights, BIOHIT may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the SOFTWARE PRODUCT in your possession.

4. COPYRIGHT
All title, including but not limited to copyrights, in and to the SOFTWARE PRODUCT and any copies thereof are owned by BIOHIT or its suppliers. All title and intellectual property rights in and to the content which may be accessed through use of the SOFTWARE PRODUCT is the property of the respective content owner and may be protected by applicable copyright or other intellectual property laws and treaties. This EULA grants you no rights to use such content. All rights not expressly granted are reserved by BIOHIT.

5. NO WARRANTIES
BIOHIT expressly disclaims any warranty for the SOFTWARE PRODUCT. The SOFTWARE PRODUCT is provided As Is without any express or implied warranty of any kind, including but not limited to any warranties of merchantability, noninfringement, or fitness of a particular purpose. BIOHIT does not warrant or assume responsibility for the accuracy or completeness of any information, text, graphics, links or other items contained within the SOFTWARE PRODUCT. BIOHIT makes no warranties respecting any harm that may be caused by the transmission of a computer virus, worm, time bomb, logic bomb, or other such computer program. BIOHIT further expressly disclaims any warranty or representation to Authorized Users or to any third party.

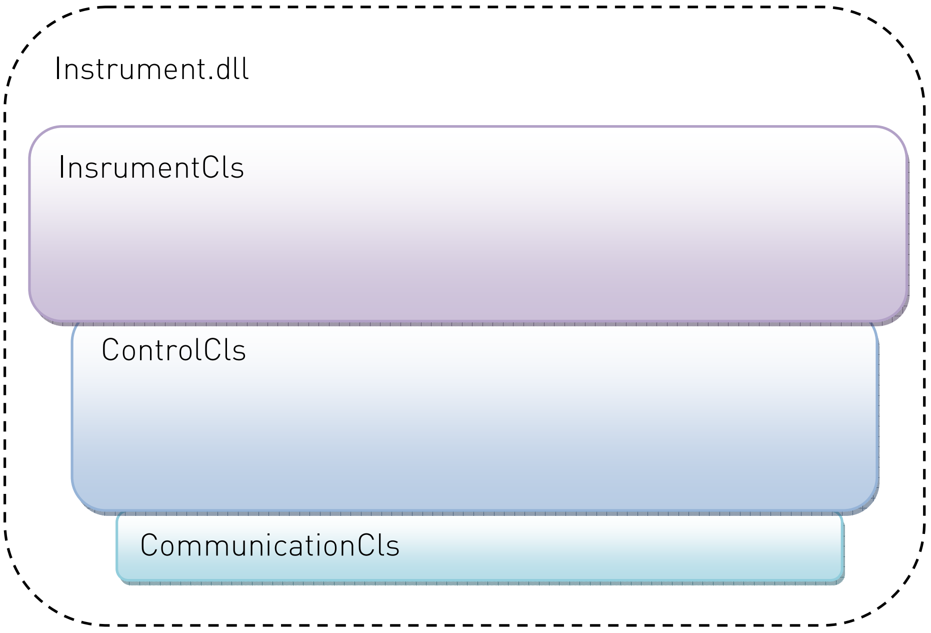
6. LIMITATION OF LIABILITY
In no event shall BIOHIT be liable for any damages (including, without limitation, lost profits, business interruption, or lost information) rising out of Authorized Users' use of or inability to use the SOFTWARE PRODUCT, even if BIOHIT has been advised of the possibility of such damages. In no event will BIOHIT be liable for loss of data or for indirect, special, incidental, consequential (including lost profit), or other damages based in contract, tort or otherwise. BIOHIT shall have no liability with respect to the content of the SOFTWARE PRODUCT or any part thereof, including but not limited to errors or omissions contained therein, libel, infringements of rights of publicity, privacy, trademark rights, business interruption, personal injury, loss of privacy, moral rights or the disclosure of confidential information.

1 Introduction

This document is intended for software designers.

It provides Application Programming Interface (API) for the Biohit Liquid Handling Automate. This API is written for the Windows OS and uses services provided by the .NET Framework 3.5. The API is provided as a Dynamic Link Library (DLL).

The InstrumentCls.dll provides three classes - **InstrumentCls** **CommunicationCls**, **ControlCls** and *VirtualCls* - with basic methods and properties to operate the Automate. VirtualCls is NOT SUPPORTED and InstrumentCls is still under development.



The syntax of the properties and methods with examples are based on C# programming language.

2 The architecture of the Automate

The Automate uses classical Master-Slave architecture what comes to the operation of the instrument, as shown in the next figure.

The master processor communicates with the Host-processor, usually with a PC, via the USB-port, and the slave processors via 2-wire differential RS485, applying asynchronous protocol.

In addition to the communication tasks, master processor will take care of some indicators (POWER, MESSAGE) and sensors (X and Y actuator position, Power measurement, Surface measurement)

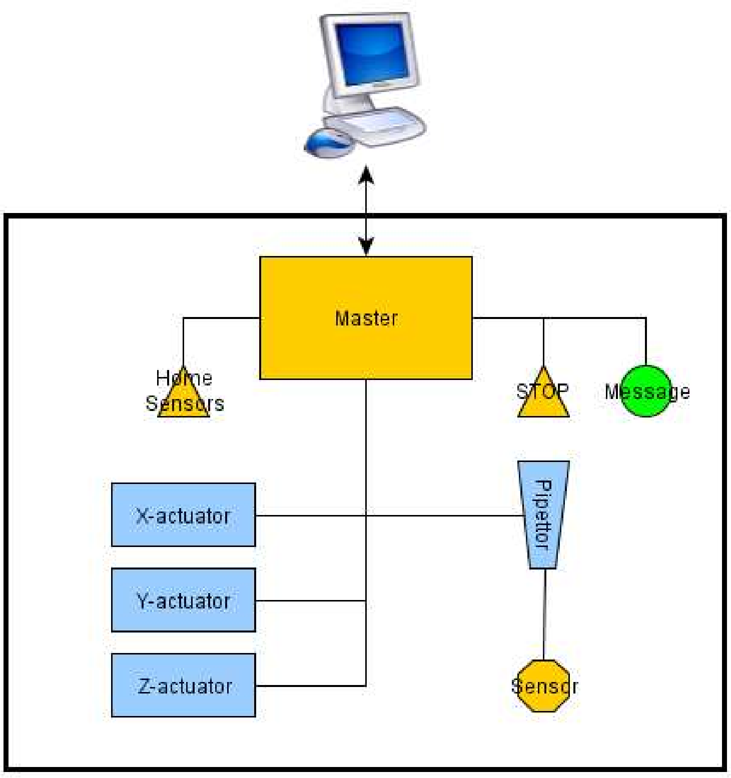


Figure 1. The architecture o f the automate.

There are four slave processors, often called as modules. The three (3) linear actuator modules (X, Y and Z) will take care of 3D movement of the Pipettor module. The Pipettor is in charge of aspirating and dispensing liquid, as well as disposing tip fixed to the pipettor tip cone.

3 API and Threading.

InstrumentCls CommunicationCls, ControlCls each of them has 2 dataloggers (error datalogger, action datalogger) total 6 dataloggers. Each datalogger has its own thread and designated folder. CommunicationCls has 2 threads one for opening communication and one for WachDog feature. InstrumentCls has one thread for auto sequence feature. So total number of thread is 9 threads.

4 Installing the USB-driver

As the instrument communication is via USB, one needs to install the corresponding USB-driver to the PC to be able to operate the automate. USB-driver is libusb/AVR32_UC3_CDC. It is provided in API installation package.

5 Using the DLL

Copy the InstrumentCls.dll to your PC and make reference to the classes. As these classes belong to the InstrumentLib-namespace, you can make an instance to these classes by using following definitions, for example.

Example 1: Create instance:

```
private InstrumentLib.InstrumentCls Instrument = new
InstrumentLib.InstrumentCls();
```

Example 2: Reference to ControlCls property:

```
Instrument.Control.PipetType = 2;
```

Example 3: How to check is USB connected from CommunicatioCls class property:

```
Instrument.Control.Comm.usbConnected==1;
```

Example 3: Close Instrument instance: When closing application YOU MUST DO!!!

```
Instrument.Dispose();
```

6 CommunicationCls

The CommunicationCls provides methods and properties to communicate with the instrument via the USB port. CommunicatioCls has a thread that automatically connects if it is possible.

To be able to use the properties and methods provided by the CommunicationCls, you must first create instance of the IntsrumentCls. InstrumenCls has a public instance of ControlCls "Control" and ControlCls has a public instance of CommunicationCls "Comm"

If the instrument is not powered on and connected to the host (PC) one can't establish communication.

6.1 Summary of the properties and methods

Summary of the properties are as follows:

ClearToReceive	- clear to receive state
usbConnected	- is usb connected
ErrorCode	- the error code
MsgIn	- last received string via the serial port
MsgOut	- last transmitted string written to the serial port
TheException	- exception information
VirtualMachine	- to enable/disable virtual machine
DataLogOnOff	- to check is data logging enabled/disabled
ErrorReceived	-ErrorRecieved

Summary of the Events are as follows:

onUsbConnect	- Fires when usb is connected
onUsbDisconnect	- Fires when usb is not connected

Summary of the methods are as follows:

IsConnected()	- Is usb connection open

SendMessage()	- send message string to the instrument
LogOnOff()	- Enables/Disables data logging

6.2 Properties

Most of the properties in this class are of type 'read only', providing information of the communication.

bool ClearToReceive

Description
ClearToReceive is set to TRUE when the message is written to the serial port. It is set to FALSE as soon as returned string is received via the serial port. As it may take some time before the reception will take place, one must avoid sending new command to the serial port if the previous message is still in process. Therefore, one option is to test the ClearToReceive before executing next command, or simply enable next transmission as soon as corresponding reception has taken place.

int usbConnected

Description
Returns the if connection is open or not.

When the connection is closed the value is false. When connection is open value is true.

int ErrorCode

Description
Returns possible communication error detected during communication.

The initial value is 0, indicating "no error''. When the port is closed the value is set to 0.

The meaning of different error codes are as follows:

0	No error- no error detected

1	Open - Failed to open serial port
2	TimeOut - Failed to receive data
3	LRC - Checksum mismatch
4	Busy - Pending message processing
9	Close - Unable to close port

string MsgIn

Description
Is the last string received via the USB-port. This information can be used to support the debugging of the API-software. It is not really needed when developing the application software.

string MsgOut

Description
Is the last message sent via the USB-port. This information can be used to support the debugging of the API-software. It is not really needed when developing the application software.

string TheException

Description
Is the error set by the .NET framework in case of method exception. This information can be used to support the debugging of the API-software. It is not really needed when developing the application software.

bool VirtualMachine

Description
Gets or Sets the so called Virtual machine operation.

The initial value is *false*, indicating 'virtual machine disabled''. If set to *true*, then all the messaging will be forwarded to the Virtual machine instead of the actual Instrument.

This feature is useful when developing and testing application software, as there is no need to have the instrument connected to the host e.g. PC.

bool DataLogOnOff

Description
Boolean value is data logging on or off.

6.3 Events

onUsbConnect (int usbConnect)

Description
Fires when usb is connected

Parameters
usbConnect usbConnect==1 then connection OK

onUsbDisconnect (int usbConnect)

Description
Fires when usb is not connected

Parameters
usbConnect usbConnect==0 then connection NOT OK

6.4 Methods

bool IsConnected ()

Description
Checks if connection is open or not.

Parameters
None.

Returns
TRUE if connected, else FALSE.

Example (C#):
bool closed = Instrument.Control.comm.IsConnected();

string **SendMessage** (string **message**)

Description
Sends message string via the USB connection.

Notes
From the application development point of view there is no need to use this method, because the ControlCls provides complete set of methods to control all the operation of the instrument. In fact, don't use this method as it may gain unexpected operation of the instrument. The method is solely used by the manufacturer when developing and debugging the API, not when developing the application software.

Parameters
message is an ASCII-string to be sent via the USB-connection.

Returns
String received message. If not successful returns `string.Empty`

Example (C#):
string receivedMsg = Instrument.Control.comm.SendMessage("1MDS");

7 ControlCls

The ControlCls provides methods and properties to control the operation of the instrument.

Also this class contains some properties and methods intended only to support manufacturer testing and debugging.

7.1 Summary of the properties and methods

Summary of the properties are as follows:

DataLogOnOff	- to check is data logging enabled/disabled
ErrorCode	- error code
PipetType	-1==200ul , 2==1000ul
screwX	-Value to scale X-axel encoder steps<->[mm]
screwY	-Value to scale Y-axel encoder steps<->[mm]
screwZ	-Value to scale Z-axel encoder steps<->[mm]
TheException	- exception information
Xcalibration	-Value off set between tray<->robot coordinate
Ycalibration	-Value off set between tray<->robot coordinate
Zcalibration	-Value off set between tray<->robot coordinate

Summary of the Events are as follows:

changePositionEvent	-If position changes, (double position, stringg address)
changeStatusEvent	-If status changes, (int status, stringg address)
changeSensorEvent	-If sensor changes, (int SensorReading)
changeErrorEvent	-If error changes, (int error, string address)
changePresentEvent	-If presence changes, (int present)

Summary of the methods are as follows:

Aspirate	- to aspirate; move piston inwards (P)

Dispense	- to dispense; move piston outwards (P)
DispenseAll	- to dispense everything out
DriveEdge	- drive actuator to the edge hard stop
DriveHome	- drive to home position
EjectTip	- eject; moves tip collar outwards (P)
Initialize	- moves actuators to zero-position (X,Y,Z,P)
WriteCalibration	- Writes calibration value of (X,Y,Z) to robot
PollCalibration	- Reads calibration value of (X,Y,Z) from robot
IsDoorInUse	- state of door installation
IsDriveOff	- state of drive off
IsDriveOn	- state of drive on
LogOnOff()	- Enables/Disables data logging
Move	- moves actuators to set position (X,Y,Z,P)
MoveToSurface	- moves until surface is detected (Z)
PickTip	- performs tip pick up (P)
PollCurrent	- returns current measurement value
PollDepth	- returns set drive depth below surface
PollDoorState	- returns door state
PollEdgePosition	- returns edge position
PollError	- returns module error register value (M,X,Y,Z,P)
PollPickUpDistance	- returns set maximum position for Tip Pick Up
PollPickUpForce	- returns pick up force value (Z)
PollPosition	- returns position of the actuator (X,Y,Z,P)
PollPresent	- returns master present register value (M)
PollSensorReading	- returns sensor reading (P)
PollSensorReference	- returns sensor triggering reference setting
PollSpeed	- returns drive in speed value (X,Y,Z,P)
PollStatus	- returns module status
PollVersion	- returns version of the instrument (M,X,Y,Z,P)
RefreshSlaves	- tests for module existence
ResetIndicators	- reset DOOR and STOP flags
ResetRegisters	- reset registers
SampleCurrent	- activates self triggered current measurement
SetBrightness	- sets the backlight brightness
SetCurrentMeasurement	- starts/stops current measurement
SetDepth	- sets drive depth below surface

SetDoor	- enables/disables door sensor
SetLRC	- enables/disables LRC-testing
SetPickUpDistance	- sets maximum position for Tip Pick Up
SetPickUpForce	- sets the tip pick up force (Z)
SetSensorReference	- sets sensor triggering reference
SetSpeed	- sets driving speed of the module (X,Y,Z,P)
Stop	- stops ongoing motion (X,Y,Z)
WaitArmToStop	- wait until arm (X and Y and Z) stops
WaitPistonToStop	- wait until piston stops

7.2 Properties

int ErrorCode

Description
Possible error detected during serial communication.

The initial value is 0, indicating "no error''. When the port is closed the value is initialized to 0.

The meaning of different error codes are as follows:

0	No error – no error detected
1	Address – invalid address
2	Parameter – invalid parameter
3	Communication – invalid or missing data received

float screwX

Description
X-actuator screw pitch as [encoder states/mm].

Notes
This value is hardcoded to the API and must match the physical actuator screw pitch for correct positioning.

float screwY

Description

Y-actuator screw pitch as [encoder states/mm].

Notes
This value is hardcoded to the API and must match the physical actuator screw pitch for correct positioning.

float screwZ

Description
Z-actuator screw pitch as [encoder states/mm].

Notes
This value is hardcoded to the API and must match the physical actuator screw pitch for correct positioning.

string TheException

Description
Is the error set by the .NET framework in case of method exception. This information can be used to support the debugging of the API-software. It is not really needed when developing the application software.

float Xcalibration

Description
X-actuator calibration value [mm]. Difference between tray and robot coordinate system [mm].

Notes
This value is calibrated to the Robot in production but it can be changed.

float Ycalibration

Description
Y-actuator calibration value [mm]. Difference between tray and robot coordinate system [mm].

Notes
This value is calibrated to the Robot in production but it can be changed.

float Zcalibration

Description
X-actuator calibration value [mm]. Difference between tray and robot coordinate system [mm].

Notes
This value is calibrated to the Robot in production but it can be changed.

bool DataLogOnOff

Description
Boolean value is data logging on or off.

7.3 Events

changePositionEvent (double position, string address)

Description
Event fires when position is asked from robot.

Parameters
position can have a value of millimeters of axel
address of axel. X,Y,Z,P

changeStatusEvent (int status, string address)

Description
Event fires when status is asked from robot.

Parameters
status can has a value of status register of address
address of unit. X,Y,Z,P,M

changeSensorEvent (int sensorReading)

Description
Event fires when sensor reading is asked from robot.

Parameters
sensorReading has a value of the sensor reading.

changePresentEvent (int **present**)

Description
Event fires when present register is asked from robot.

Parameters
present has a value of the present register.

7.4 Methods

Please notice that many of the methods in this class support optional **wait** parameter. The purpose of this parameter is define when to return from the method; either as soon as the drive has been started (FALSE) or completed (TRUE). This feature, however, is protected by a 10 second Time Out – operation.

bool **Aspirate** (float **volume**, *optional* bool **steps**, *optional* bool **wait**)

Description
Aspirates set volume.

Parameters
volume can have a value of microliters or steps.
steps defines the unit – microliters (FALSE) or steps (TRUE)
wait defines return from the function; if TRUE, the return will take place as soon as the aspiration is fully completed, if FALSE the return takes place as soon as the command has been sent to the instrument.

Returns
TRUE is returned when execution is completed, else FALSE. If wait is missing or FALSE return of TRUE indicates accepted command. If wait is set to TRUE with FALSE return value, one must check for the error and continue accordingly.

Notes
It may take even a few seconds before function returns, especially if the piston speed is low and high volume are to be aspirated.

Example (C#):
// starts to aspirate 100 ul
bool status = Instrument.Control.Aspirate(100.0f);

bool **Dispense** (float **volume**, *optional* bool **steps**, *optional* bool **wait**)

Description
Dispenses set volume.

Parameters
volume can have a value of microliters or steps.
steps defines the unit – microliters (FALSE) or steps (TRUE)
wait defines return from the function; if TRUE, then return will take place as soon as the dispense is completed, if FALSE the return takes place when the command has been sent to the instrument.

Returns
TRUE is returned when execution is completed, else FALSE. If wait is missing or FALSE return of TRUE indicates accepted command. If wait is set to TRUE with FALSE return value, one must check for the error and continue accordingly.

Notes
It may take even a few seconds before function returns, especially if the piston speed is low and high volume are to be dispensed.

Example (C#):
// dispenses 50 ul
bool status = Instrument.Control.Dispense(50.0f, false, true);

bool **DispenseAll** (*optional* bool **wait**)

Description
Dispenses everything out from the tip.

Parameters
wait defines return from the function; if TRUE, then return will take place as soon as the aspiration is completed, if FALSE the return takes place when the command has been sent to the instrument.

Returns
TRUE is returned when execution is completed, else FALSE. If wait is missing or FALSE return of TRUE indicates accepted command. If wait is set to TRUE with FALSE return value, one must check for the error and continue accordingly.

Notes
It may take few seconds to complete the operation with returned value.

Example (C#):
// dispenses all
bool status = Instrument.Control.DispenseAll();

bool **DriveEdge** (string **address**, *optional* bool **wait**)

Description
Drives the actuator to mechanical hard stop.

Parameters
address of the module "X" or "Y" or "Z".
wait defines return from the function; if TRUE, then return will take place as soon as the motion is completed, if FALSE the return takes place when the command has been sent to the instrument.

Returns
TRUE is returned when execution is completed, else FALSE. If wait is missing or FALSE return of TRUE indicates accepted command. If wait is set to TRUE with FALSE return value, one must check for the error and continue accordingly.

Notes
This method is intended for service use, only. There is no need to use this with application development.

Example (C#):
bool status = DriveEdge();

bool **DriveHome** (string **address**, *optional* bool **wait**)

Description
Drives actuator to home-position.

Parameters
address of the module "X" or "Y" or "Z".
wait defines return from the function; if TRUE, then return will take place as soon as the motion is completed, if FALSE the return takes place when the command has been sent to the instrument.

Returns
TRUE is returned when execution is completed, else FALSE. If wait is missing or FALSE return of TRUE indicates accepted command. If wait is set to TRUE with FALSE return value, one must check for the error and continue accordingly.

Notes
DriveHome() must be successfully executed after instrument has been powered on.
Make sure that the Z-actuator is homed first before executing this function to X or Y. This way you avoid possible crashes against the lab ware placed on the tray. If you need to Home all the modules, it is recommended to use the Initialize()-method, to take care of correct sequential order of all the actions.

Example (C#):
bool status = Instrument.Control.DriveHome();

bool EjectTip (*optional* bool wait)

Description
Ejects tip.

Parameters
wait defines return from the function; if TRUE, then return will take place as soon as the motion is completed, if FALSE the return takes place when the command has been sent to the instrument.

Returns
TRUE is returned when execution is completed, else FALSE. If wait is missing or FALSE return of TRUE indicates accepted command. If wait is set to TRUE with FALSE return value, one must check for the error and continue accordingly.

Notes
It may take few seconds, if wait is set to TRUE, to complete the operation with returned value.

Example (C#):
// start tip ejecting
bool status = EjectTip();

bool **Initialize** (*optional* string **address**)

Description
Energizes module (X|Y|Z|P) motors to drive to the 0-position. All the actuators will become initialized.

Parameters
address of the module "X" or "Y" or "Z" or "P".

Returns
TRUE, if the initialization was successful, else FALSE.

Notes
This function activates driving the modules until 0-position is reached. The position of the actuator is 0 after the execution.

Example (C#):
// runs all the actuators to hardware home position
bool status = Instrument.Control.Initialize();

int **IsDoorInUse** ()

Description
Door state of the instrument.

Parameters
None.

Returns
If the door is in use (and sensor is enabled), then the return value is TRUE. If the door sensor is disabled or communication with the instrument fails, the

returned value is FALSE. Therefore, true disabled door sensor states requires none error.

Example (C#):
int status = Instrument.Control.IsDoorInUse();

bool **IsDriveOff** (string **address**)

Description
Tests if module drive is off.

Parameters
address of the module "X" or "Y" or "Z" or "P".

Returns
TRUE if module is not in motion, else FALSE.

Notes
FALSE is returned if drive is on or communication fails. Therefore to distinguish between these two options, it is recommended to check the error-status after returned FALSE. If there is no error with communication, then the drive is on.

Example (C#):
bool status = Instrument.Control.IsDriveOff("Y");

bool **IsDriveOn** (string **address**)

Description
Tests if module drive is on.

Parameters
address of the module "X" or "Y" or "Z" or "P".

Returns
TRUE if module motion is active, else FALSE.

Notes
FALSE is returned if drive is off or communication fails. Therefore to distinguish between these two options, it is recommended to check the

error-status after returned FALSE. If there is no error with communication, then the drive is off.

Example (C#):
bool status = Instrument.Control.IsDriveOn("X");

bool **Move** (string **address**, float **position**, *optional* bool **steps**, *optional* bool **wait**)

Description
Moves the actuator of the modules (X|Y|Z|P) to the set position.

Parameters
address of the module "X" or "Y" or "Z" or "P".
Target **position**.
Displacement **steps** is FALSE for mm (X|Y|Z) / nanoliter (P), or TRUE for steps.
wait defines return from the function; if TRUE, then return will take place as soon as the motion is completed, if FALSE the return takes place when the command has been sent to the instrument.

Returns
TRUE is returned when execution is completed, else FALSE. If wait is missing or FALSE, then TRUE as return value indicates accepted command. If wait is set to TRUE with FALSE return value, one must check for the error and continue accordingly.

Notes
If **steps** is true, the position should be given as integer as it should represent value of steps. If the value is given as decimal it will be rounded to integer before executing the drive.

Example (C#):
// start Y-actuator movement towards position 23.6 mm
bool status = Instrument.Control.Move("Y", 23.6);

bool **MoveToSurface** (float **limit** , *optional* bool **wait**)

Description
Moves downwards (Z) until surface has been detected.

Parameters
The limit defines maximum position to go.
wait defines return from the function; if TRUE, then return will take place as soon as the motion is completed, if FALSE the return takes place when the command has been sent to the instrument.

Returns
TRUE, if the execution was successful (= surface was detected), else FALSE (=surface was not detected).

Notes
If the initial position is 'above' the limit, then search will proceed downwards, until tip is touching the liquid surface.
If the initial position is 'below' the limit, then search will proceed upwards, until tip has been lifted out from the liquid.

Downward direction: If surface is detected, movement is stopped, tip below the liquid surface. If surface was not detected, the movement is stopped close to limit position. In practice, the limit position must be set above the bottom of the container to not touch the container bottom.

Upward direction: If surface is detected, movement is stopped, tip just above the liquid surface. If surface was not detected, the movement is stopped close to the limit. In practice, the limit position can be set above the container.

Example (C#):
// start Y-actuator movement downwards (if current position> 31.0), if not found, stops at 31 mm
bool status = Instrument.Control.MoveToSurface(31.0f);

bool PickTip (*optional* bool wait)

Description
Picks up tip.

Parameters
wait defines return from the function; if TRUE, then return will take place as soon as the motion is completed, if FALSE the return takes place when the command has been sent to the instrument.

Returns
TRUE is returned when execution is successfully over, else FALSE. If FALSE is returned one needs to figure out the cause. It can be either communication error or unable to dash against the tip at expected position.

Notes
It may take even a few seconds before function returns, in case the wait is set to TRUE.

Example (C#):
// picks tip up
bool status = Instrument.Control.PickTip(TRUE);

float **PollCalibration** (string address)

Description
Polls the calibration value of address in millimeters.

Parameters
Address defines the address value. X,Y,Z

Returns
Calibration value of address in millimeters or -1 in case of communication failure.

Example (C#):
// Read X-axels calibration value from robots Master unit.
float Xcalibration= Instrument.Control.PollCalibration("X");
…

int **PollCurrent** (bool **triggered**)

Description
Polls the last current value measured in milliamperes.

Parameters
Triggered defines the type of the current measurement; either last self triggered 100 ms average or last instant value.

Returns
Last current value as positive integer, or -1 in case of communication failure.

Notes
Triggered measurement (SampleCurrent()) requires special conditions and is therefore useful only when the speed is high 8 or 9. With lower speeds one must use the non-triggered measurement (SetCurrentMeasurement()). Please refer to the corresponding methods SetCurrentMeasurement() and SampleCurrent() to activate current measurement.

This method is intended for service use only. There is no need to use this with application software.

Example (C#):
```
// Start self-triggered current measurement
bool status = Instrument.Control.SampleCurrent();
...
// Start run with high speed
...
// Return the latest value
int current = PollCurrent( true );
```

float **PollDepth** ()

Description
Polls depth-below-surface setting.

Parameters
None

Returns
Depth setting (in steps) as positive integer, or -1 in case of communication failure.

Example (C#):
```
float depth = Instrument.Control.PollDepth();
```

int **PollDoorState** ()

Description
Polls door state; open or close.

Parameters
None

Returns
Door state; 1 = open, 0 = close, -1 in case of communication failure.

Example (C#):
int status = Instrument.Control.PollDoorStatus();

int **PollEdgePosition** (string **address**)

Description
Real zero position of the actuator.

Parameters
address of the module "X" or "Y" or "Z" or "P".

Returns
Zero-position (in steps) of the actuator after successful DriveEdge-execution.

Notes
This method is for service use only. There is no need to use it with application development.

Example (C#):
int position = Instrument.Control.PollEdgePosition("Z");

int **PollError** (string **address**)

Description
Returns error register value of the module (M|X|Y|Z|P)

Parameters
Address of the module "M" or "X" or "Y" or "Z" or "P".

Returns
Status of the module, or -1 if not able to access valid value.

Notes
Master (M) register holds the module level error information of the instrument. It is the primary register to poll the system error. Detailed module error information is provided by polling the corresponding module by address.

Please refer to the chapter "The Registers" for more information about the error registers.

Example (C#):
int error = Instrument.Control.PollError("M");

float PollPickUpDistance ()

Description
Polls maximum tip pick up position setting.

Parameters
None

Returns
Pick up position setting as positive integer, or -1 in case of communication failure.

Notes
Only for special use.

Example (C#):
float max = Instrument.Control.PollPickUpDistance();

int PollPickUpForce ()

Description
Returns current tip pick up force value of the Z-module.

Parameters

None

Returns
Tip pick up force, or -1 if not able to access valid value.

Notes
Only for special use.

Example (C#):
// returns tip pick up force
int force = Instrument.Control.PollPickUpForce();

float **PollPosition** (string **address**, *optional* bool **steps**)

Description
Returns position of the actuator of the module (X|Y|Z|P)

Parameters
address of the module "X" or "Y" or "Z" or "P".
If **steps** is TRUE, then position is returned in steps, else in millimeters or microliters.

Returns
Position of the module, or -1000 if not able to access valid value.

Notes
If steps is true, the value returned is steps for all of the modules. If the steps is false, or missing, the value returned value is mm for X|Y|Z, but microliters with P.

Example (C#):
// returns X-position in mm's
float position = Instrument.Control.PollPosition("X");

int **PollPresent** ()

Description
Returns present register value of the Master module (M)

Parameters

None.

Returns
Presence register value of the modules, or -1 if not able to access valid value.

Notes
Please refer to the chapter "The Registers" for more information about the registers.

Example (C#):
int present = Instrument.Control.PollPresent("M");

int PollSensorReading (*optional* string address)

Description
Returns sensor reading. The method returns the value of the sensor surface from master counting a predetermined number of oscillation. The frequency varies depending on the permittivity material. Frequency range of 526 ... 100 Hz is corresponding sensor reading range of about 11 400 ... 60000.

Parameters
None

Returns
Sensor reading, or -1 if not able to access valid value.

Notes
The liquid surface sensor is located at the top of the pipettor tip cone. The operation is supported only when conductive tips are used.

Example (C#):
float reading = Instrument.Control.PollSensor();

int PollSensorReference ()

Description
Returns current sensor triggering reference.

Parameters

None

Returns
Sensor triggering reference value, or -1 if not able to access valid value.

Notes
Only for special use.

Example (C#):
// returns Sensor Sensitivity
int sensitivity = Instrument.Control.PollSensorSensitivity();

int PollSpeed (string address, *optional* bool inwards)

Description
Returns current set speed value of the module (X|Y|Z|P).

Parameters
address of the module "X" or "Y" or "Z" or "P".
inwards defines the direction of the movement.

Returns
Speed value, or -1 if not able to access valid value.

Notes
For X|Y|Z speed is the same for both directions and therefore inwards is meaningless – it can be omitted.
For Pipettor the direction must be defined – TRUE for inwards speed and FALSE for outwards speed.

Example (C#):
// returns Pipettor dispensing speed
int speedOut = Instrument.Control.PollSpeed("P", true);

int PollStatus (string address)

Description
Returns status register value of the module (M|X|Y|Z|P)

Parameters

address of the module "M" or "X" or "Y" or "Z" or "P".

Returns
Status register value of the module, or -1 if not able to access valid value.

Notes
Master (M) register holds the general status of the whole instrument. It is the primary register to poll the system status. The other registers are for service use only.

Please refer to the chapter "Registers" for more information about the status register.

Example (C#):
```
int status = Instrument.Control.PollStatus("M");
```

string **PollVersion** (string **address**)

Description
Returns version of the module (M|X|Y|Z|P)

Parameters
address of the module "M" or "X" or "Y" or "Z" or "P".

Returns
Software version (as string) of the module

Example (C#):
```
string version = Instrument.Control.PollVersion();
```

bool **RefreshSlaves** ()

Description
Tests module presence.

Parameters
None.

Returns

TRUE if properly executed, else FALSE.

Notes
If any of the modules is not able to communicate, module presence flag may become disabled. Should this happen, RefreshSlaves() can be executed to try to refresh slaves. Status of the present register indicates the result.

Please refer to the chapter "Registers" for more information about the status register.

Example (C#):
```
bool status = Instrument.Control.RefreshSlaves();
```

bool ResetIndicators ()

Description
Resets drive disabling DOOR and STOP –flags.

Parameters
None.

Returns
TRUE if properly executed, else FALSE.

Notes
If door is opened or STOP-button pressed, corresponding DOOR or STOP-flag is set. If any of these flags are set, it is not possible to drive the instrument. Therefore, one must execute ResetIndicators()-method, to reset these flags, before next drive execution.

If the door is not used and the corresponding door sensor is disabled, the door flag is not set.

Example (C#):
```
bool status = Instrument.Control.ResetIndicators();
```

bool ResetRegisters ()

Description
Resets instrument registers.

Parameters
None

Returns
TRUE if properly executed, else FALSE.

Notes
Only for special use.

Example (C#):
```
bool status = Instrument.Control.ResetRegisters();
```

bool **SampleCurrent** ()

Description
Activates self triggered current measurement.

Parameters
None.

Returns
TRUE if the measurement was properly initiated, else FALSE.

Notes
This method activates current measurement, and triggers 100 ms current value integration as soon as high current consumption takes place. Therefore, it is important to initiate reasonable acceleration before triggering can take place. In practice, this feature requires high speed setting, either 8 or 9.

The PollCurrent(TRUE)-method returns the last measured value.

Current measurement is used for testing purposes, for example, to indicate the service needed of the actuators.

Example (C#):
```
// Start self-triggered current measurement
bool status = Instrument.Control.SampleCurrent();
```

```
...
// Start run with high speed
...
// Return the latest value
int current = Instrument.Control.PollCurrent( true );
```

bool **SetBrightness** (int **brightness**)

Description
Sets the brightness of the instrument back panel.

Parameters
brightness defines the luminous intensity of the back panel. Range from 0 to 100, default value is 0.

Returns
TRUE if the brightness was properly set, else FALSE.

Notes
Set the brightness value to 0 to switch the back light off.

Example (C#):

```
// sets brightness to maximum intensity
bool status = SetBacklight(100);
```

bool **SetCurrentMeasurement** (bool **on**)

Description
Activates current measurement.

Parameters
on activate/inactivate the current measurement. As default, the current measurement is off (on = false).

Returns
TRUE if properly executed, else FALSE.

Notes
This method activates/inactivate current measurement. The sampling rate is about 130 Hz. The instant value can be polled at any time.

This method is recommended when low motor speed is used. With high speed, use the SampleCurrent()-method.

Continuous current measurement consumes processor resources and is therefore recommended to be turned off when current measurement is not really need.

Current measurement is used for testing purposes, for example, to indicate the service needed of the actuators.

Example (C#):

```
// Set continuous current measurement on
bool status = SetCurrentMeasurement(true);
...
// Start run
...
// Read current value at plateau phase
Int current = Instrument.Control.PollCurrent();
```

bool **SetDepth** (float **distance**)

Description

Sets depth-below-surface distance.

Parameters

distance in millimeters from detected surface. Range from 0 to 9.9 mm, default value is 0.

Returns

TRUE if properly executed, else FALSE.

Example (C#):

```
bool status = Instrument.Control.SetDepth(3.0f);
```

bool **SetDoor** (bool **use**)

Description

Enables/disables DOOR-sensor.

Parameters
If door is used and enabled, use must be set to TRUE. If door is removed, use must be set to FALSE. As default the door is enabled in use.

Returns
TRUE if properly executed, else FALSE.

Notes
Even thought the door is installed, the corresponding DOOR-sensor can be disabled by executing SetDoor(false), to support testing/debugging the operation when developing application software.

Example (C#):
bool status = Instrument.Control.SetDoor(false);

bool SetLRC (bool on)

Description
Enables/disables LRC-testing.

Parameters
To enable LRC-testing, on must be set to TRUE. To disable LRC-testing, on must be set to FALSE. As default the LRC-testing is disabled in use.

Returns
TRUE if properly executed, else FALSE.

Notes
All the messages carry LRC-byte calculated from the message string. However, from testing point of view it is more practical to communicate the instrument if correct LRC-is not required. This can be achieved by using the SetLRC(false) –method.

If the LRC-testing is required, the corresponding communication method automatically resends a message if the automate replies with incorrect LRC. Also, if communication method sends a message with incorrect LRC, the automate will not process the message if LRC-testing is enabled.

Even if LRC-testing is enabled, it can be disabled with SetLRC(false) method.

Example (C#):

```
bool status = Instrument.Control.SetLRC(true);
```

bool **SetPickUpDistance** (float **position**)

Description
Sets the maximum position used with PickTip()-method.

Parameters
Maximum (minimum z-axis position) **position** in steps devided by 100. The valid values are between 50 to 99. The default value is 79.

Notes
For special use only.

Returns
TRUE if properly executed, else FALSE.

Notes
If the maximum position is reached, then an error is raised. When properly set the raised error can be used for tip pick testing.

Example (C#):
```
bool status = Instrument.Control.SetPickUpDistance(80);
```

bool **SetPickUpForce** (int **force**)

Description
Sets the tip pick up force (Z).

Parameters
force is a positive integer, between 8 to 26. The default value is 10.

Returns
TRUE if the force value was properly set, else FALSE.

Notes
For special use only.

Example (C#):
```
// sets pick up force
```

bool status = Instrument.Control.SetPickUpForce(15);

bool **SetSensorReference** (float **reference**)

Description
Sets liquid surface sensor triggering reference.

Parameters
Reference is a positive float. The value corresponds to a percentage (%) triggering change by a formula ½^(12-reference). Reference can have values between 0 to 12. The default value is 5, corresponding to 0.78% change in reading.

Returns
TRUE if the force value was properly set, else FALSE.

Example (C#):
bool status = Instrument.Control.SetSensorReference(4);

bool **SetSpeed** (string **address**, int **speed**, *optional* bool **inwards**)

Description
Sets the current speed value for the module (X|Y|Z), for selected direction (P).

Parameters
address of the module "X" or "Y" or "Z" or "P".
speed is the speed setting from 1-6 (P) or 1-9 (X|Y|Z). The default values are as follows: P:3. X,Y: 8, Z:9.
inwards defines the direction of the movement.

Returns
TRUE if the speed was properly set, else FALSE.

Notes
For X|Y|Z speed is the same for both directions and therefore inwards is meaningless – it can be omitted.
For Pipettor the direction must be defined – TRUE for inwards speed and FALSE for outwards speed.

Example (C#):

```
// sets aspiration speed to 2
bool status = Instrument.Control.SetSpeed("P",2, true);

// sets dispensing speed to 6
bool status = Instrument.Control.SetSpeed("P",6,false);
```

bool **Stop** (optional string **address**)

Description
Stops on-going movement of the modules (X|Y|Z).

Parameters
address of the module "X" or "Y" or "Z".

Returns
TRUE, if the execution was successful, else FALSE.

Notes
This function is used for emergency stop purposes.

Example (C#):

```
// stops all the actuators (X,Y,Z)
bool status = Instrument.Control.Stop();
```

bool **WaitArmToStop** (string **address**)

Description
Waits until actuator movement is over.

Parameters
Address of the module "X" or "Y" or "Z".

Returns
TRUE is returned when process is completed successfully, else FALSE.

Notes
It may take a while before function returns a value, especially if drive speed is low or distance to travel is long.

This method is protected by TimeOut – functionality.

Example (C#):
// waits until X-motion is completed
bool status = Instrument.Control.WaitActuatorToStop("X");

bool **WaitArmToStop** ()

Description
Waits until 3D-robotic arm movement is over.

Parameters
None.

Returns
TRUE is returned when process is completed successfully, else FALSE.

Notes
It may take a while before function returns a value, especially if drive speeds are low or distances to travel are long.

This method is protected by TimeOut – functionality.

Example (C#):
// waits until X-motion has finished
bool status = Instrument.Control.WaitArmToStop();

bool **WaitPistonToStop** ( )

Description
Waits until piston movement is over.

Parameters
None.

Returns
TRUE is returned when process is completed successfully, else FALSE.

Notes
It may take a while before function returns a value, especially if drive speed is low or distance to travel is long.

This method is protected by TimeOut – functionality.

Example (C#):
// waits until X-motion has finished
bool status = Instrument.Control.WaitPistonToStop();

bool **WriteCalibration** (string **address , float mCalibration**)

Description
Writes calibration value to robots master unit.

Parameters
Address. Address of calibration value. mCalibration, value of calibration in millimeters.

Returns
TRUE is returned write ok, else FALSE.

Example (C#):
// waits until X-motion has finished
bool status = Instrument.Control.WriteCalibration("X", 1.0);

7 InstrumentCls

InstrumentCls is still under development.

8.1 Summary of the properties and methods

Summary of the properties are as follows:

TheError	- last error

TheException	- exception information
DataLogOnOff	- to check is data logging enabled/disabled
X_position	-Position of X-axel
Y_position	-Position of Y-axel
Z_position	-Position of Z-axel
P_position	-Position of pipette

Summary of the Events are as follows:

sequenceStatusChanged	-If sequence status, (int row, int status)
sequenceStoppedEvent	- If sequence stopped, (String message, int Stopped)
sequenceErrorEvent	-If sequence error, (String ErrorMsg, int ErrorNum, int Pause, int Stopped)

Summary of the methods are as follows:

VirtualMachine	-is virtual machine in use
SetVirtualMachine	-set virtual machine in use/not use
IsConnected	- Is the USB -connection to robot ok
SendMessage	-Send message to Robot
LogOnOff	- Enable/Disable data logging
InitializeInstrument	- Initialize instrument
ResetIndicators	- Reset indicators
SetBrightness	- Sets Brightness
MoveXY	- Move X-/Y-axel to position
MoveZ	- Move Z-axel to position
MoveToSurface	-Move to surface
Aspirate	-Aspirate
Dispense	-Dispense
DispenseAll	- Dispense all
MovePistonToPosition	- Move piston to position
SetAspirateSpeed	- Set aspirate speed
SetDispenseSpeed	- Set dispense speed
SetActuatorSpeed	- Set actuator speed
SetPickUpForce	- Set pick up force
SetPickUpDistance	- Set pick up distance

GetPickUpDistance	- Get pick up distance
SetSensorReference	- Set Sensor Reference
IsDoorInUse	- Gets if door is in use
SetDoor	- Set door to use/not use
WriteCalibration	- Write calibration to robots master card
PickTip	- Pick tip
EjectTip	- Eject tip
getSequenceTable	- Gets sequence DataTable of sequence machine
setSequenceTable	- Set s sequence DataTable
pauseSequence	- Pause sequence
continueSequence	- After pause continue sequence
stopSequence	- Stop sequence
startSequence	- Start sequence

9 Registers

9.1 Introduction

The system provides status information packed via status, present and error registers. Each of these registers are actually holding 8-bit data. Each bit is dedicated to certain purpose.

Each of the register values can be polled with one call making system polling fast, as being essential because of the asynchronous nature.

The Master ("M") register is located in the master processor and is therefore the primary source of status information. The slave registers are located in the slave processors and are used when detailed information of the error is needed.

Notice that the Master register access time is much shorter that slave registers. Therefore it is essential to trace the possible error with hierarchical manner. Please refer to the "Tracing the Error"- chapter on this subject.

9.2 Present register

The Master Present register holds information of the presence of different modules – X, Y, Z actuators and the Pipettor.

The Present register bits are as follows:

8	7	6	5	4	3	2	1
		DOOR	P	Z	Y	X	M

Table 7.1. Present register bits.

The Least Significant Bit **M** is set when initialization has been successfully carried out.

X, Y, Z, P are set when modules have been properly initialized.

The **DOOR** bit is set after start up, independent if the door is used or not. If the DOOR is not used, this bit can be cleared by the SetDoor()-method.

After power on the Preset register value is 63. If any of the modules X,Y,Z or P are not present – communicating with the master – then the corresponding bit is not set. Should this happen, at start up or during operation, it indicates a fatal error and needs immediate solving.

9.3 Master Status register

The Master Status register holds information of the status of the different modules – X, Y, Z actuators, Pipettor, door, pause and tray.

The Status register bits are as follows:

8	7	6	5	4	3	2	1
TRAY	PAUSE	DOOR	P	Z	Y	X	GE

Table 7.2. Master Status register bits.

GE is set if general error has been generated and operation need further checking.

X, Y, Z, P are set as long as corresponding drive execution is on.

DOOR is set when the DOOR is opened.

PAUSE is set when Pause button has been pressed.

TRAY (reserved for the TRAY use).

If PAUSE or DOOR are set during operation with the instrument, one must clear these bit programmatically to be able to continue operation. By this way the state change interrupt generated will become confirmed. The ResetIndicators()-method is intended for clearing these bits.

Notice

Even though there exist also module level status registers these are useless what comes to the application development, and are therefore omittied from this manual.

9.4 Error registers

Both the Master as well as each of the modules provides error registers for internal error information.

9.4.1 The Master Error Register

The Master error provides module level error information. It contains following bits:

8	7	6	5	4	3	2	1
			P	Z	Y	X	M

Table 7.3. Master Error register bits.

M is set if general error has been generated.

X, Y, Z, P are set if an error has been detected in any of these modules. Should this happen, more detailed information about the error can be figured out by polling the X,Y,Z and P-module error registers.

9.4.2 The Slave Error Registers

The actuator error register contains following bits:

8	7	6	5	4	3	2	1
Reset				Tip	Home	Over	Jam

Table 7.4. Slave Error register bits.

Jam is set if the module is not able to reach the set position.

Over is set if the final position is not within the predefined limits.

Home is set if the Initialization() fails; instrument needs to be serviced.

Tip is set – only with Z-module – if the PickTip()-method fails to dash against the tip.

Notice – the Pipettor Error register is not yet documented in this manual.

9.5 Tracing the Error

If any of the methods used to drive the actuators X, Y, Z or the pipettor P returns FALSE, one must take actions to figure out the cause.

If the error is not due to a communication, the first thing to do is to poll the status of the instrument with the help of PollStatus("M") – method. If the M-bit is set, then instrument error has been detected. Next we need to figure out which of the module did raise the error, by executing PollError("M") – method. The return value provides information of the module in question. And, finally to figure out the cause of the error, one needs to execute the same method with the corresponding module address as parameter.

Example: PickTip() method returns FALSE.

- No communication error was detected, which triggers to figure out instrument error.
- Return value of PollStatus("M") is 1, which triggers to execute method PollError().
- Return value of PollError("M") is 8, indicating error with Z-module.
- Return value of PollError("Z") is 8, indicating TIP-bit set. The propable cause is a missing tip from the tip tray position.