Prior Scientific Instruments

DLL API Description & Command Set

Document Number:

Version	Changes
1.0.0	Initial version
1.0.1	Adding ODS loader functions
1.0.2	Adding shutter and filter functions Adding logging functions
1.1.0	Adding API for SL160 ,TTL, LED, OEM and controller wide functions
1.2.0	Adding API for WASLV2
1.3.0	Adding API for stage/z backlash

CONTENTS

1. Introduction

Prior controllers (ProScan3 and OptiScan3) allow software control of high precision stages (X,Y and Z) in both open and closed loop with stepper or linear motor control. Other connected ancillary equipment can also be also be controlled, such as filter wheels, shutters, LED’s and other generic OEM stepper motor controlled devices.

Along with this controller, other equipment can be ‘paired’ with to provide automated handling of slides and well plates for example. These devices are generically called ‘loaders’ and their purpose is to select a slide or plate from a hotel and place that on the stage ready for user’s scanning application to process.

1.1. DLL and Application Programming Interface

The DLL has a simple function-based application programming interface (API) implemented as a standard Windows “C” DLL. Applications importing the DLL library can be written directly in any programming language such as C, C++, C#, Python etc, or may be designed in environments such as Matlab or Labview.

The interface comprises of five functions:

1.1.1. Version

*int PriorScientificSDK_Version(char * const version);*

The DLL version number is returned via the *version* pointer. This pointer must be valid and point to a buffer large enough to hold the returned null terminated ASCII string (suggested minimum size of 20 bytes). The returned string is in the format “x.y.z” conforming to *major, minor, patch* semantic versioning notation.

1.1.2. Initialisation

int PriorScientificSDK_Initialise(void);

Before using any other DLL API (other than version) it is necessary for the DLL to configure its internal data structures.

1.1.3. Session Management

The DLL is capable of supporting multiple sessions. A session consists of a connection to a controller (ProScan3 or optiScan3) with associated connected ancillary devices plus also an associated loader (or other future devices). Internally these devices are logically connected and aware of each other. Controllers/devices on different sessions are unaware of each other.

Currently this is limited to 10 but may change in the future.

1.1.3.1. Session Open

```
int PriorScientificSDK_OpenNewSession(void);
```

Creates a new session and all the required data objects to go with it, returning a non-negative session identifier which should be used when sending commands or closing the session.

1.1.3.2. Session Close

```
int PriorScientificSDK_CloseSession(int sessionID);
```

Close the open session specified by sessionID. This destroys all the data objects currently associated with this session. It should be the last function called during any open session.

1.1.4. Commands

```
int PriorScientificSDK_cmd(int sessionID, const  char * const cmd, char * const  result);
```

All commands are passed using this function and are ASCII null terminated strings. Internally in the DLL they are truncated to a maximum length of 256 bytes. The calling application is responsible for providing a valid pointer to a buffer of minimum 512 bytes into which a NULL terminated ASCII command result will be written. The result string is only valid if the function response is PRIOR_OK.

```
int apiError;

char result[512];

apiError = PriorScientificSDK_cmd(sessionID, “controller.stage.position.get”, result);
```

1.2. API Error codes

Refer to section *SDK Error Codes*

1.3. Command Format

All commands consist of two parts: an initial command followed by an optional space separated parameter list.

The command string follows a structured approach identifying main controller, sub device, property or function name. Technically, there is no difference between the two variants, property or function, it is just simple nomenclature style designed to be memorable.

There is no locale conversions available for the command strings, they must be written exactly in lower-case English as described below. Responses also will be in English-Great Britain language-region format. If need be the user should convert into local language, for say, display purposes.

1.3.1. Properties

"controller.stage.position.get" identifies main *controller* (could be ProScan or OptiScan), a sub device of *stage* and a property *position*. Properties have a *get* and *set* operation associated with them and may take parameters.

1.3.2. Methods

"controller.stage.goto-position 1234 5678" identifies main *controller* (could be ProScan or OptiScan), a sub device of *stage* and a function *goto-position* which takes two parameters and X and Y stage position

1.4. String Parameters

All string parameters are ordinary null terminated C-style strings.

If using the DLL in managed environment such as C# then the interface should be imported in the following way.

```
[DllImport("PriorScientificSDK.dll", CallingConvention = CallingConvention.Cdecl)]
        public static extern int PriorScientificSDK_Version(StringBuilder version);

[DllImport("PriorScientificSDK.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern int PriorScientificSDK_Initialise();

[DllImport("PriorScientificSDK.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern int PriorScientificSDK_OpenNewSession();

[DllImport("PriorScientificSDK.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern int PriorScientificSDK_CloseSession(int sessionID);

[DllImport("PriorScientificSDK.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern int PriorScientificSDK_cmd(int session, StringBuilder tx, StringBuilder rx);
```

These DLL entry points can be used as-is, or abstracted into a C# class specification.

2. Logging

2.1. Log Path

Description	Logging information from the dll will be written to PriorSDK.log file in the specified folder when logging enabled. If this command not called then the log path defaults to the folder the dll exists in.
Command	`dll.log.path <path>`
Parameters	`<path>` — A fully qualified path e.g `"dll.log.path C:\\Users\\fred\\Desktop”` — *string*
Result	“0”

2.2. Logging on

Description	Turn logging on.
Command	`dll.log.on`
Parameters	none
Result	“0”

2.3. Logging off

Description	Turn logging off.
Command	`Dll.log.off`
Parameters	none
Result	“0”

3. Controller Commands

In all cases if the *PriorScientificSDK_cmd* returns *PRIOR_OK* status then the *result* parameter string contains the response from the controller. All numbers are returned as string equivalent values and should be converted by the user application.

If the command returns PRIOR_CONTROLLERERROR then the *controller.lasterror.get* can be used to determine the controller specific error code. See *Controller Error Codes*

3.1. System Level Commands

3.1.1. controller.connect

Description	Establish a communications connection between the DLL and the controller on the specified port.		
Command	`controller.connect <port>`		
Parameters	`<port>`	The numerical value of the port as described in the device manager. I.e. for “COM3” use the value “3”	*int*
Result	“0”		

3.1.2. controller.disconnect

Description	Closes the currently open communications channel to the controller.
Command	`controller.disconnect`
Parameters	None
Result	“0”

3.1.3. controller.lasterror.get

Description	Returns the last
Command	`controller.lasterror.get`
Parameters	None
Result	Last error code

3.1.4. controller.stop.smoothly

Description	Stops all axes moving in a controlled fashion, following the acceleration and jerk settings for each axis. Positional accuracy is maintained.
Command	`controller.stop.smoothly`
Parameters	None
Result	“0”

3.1.5. controller.stop.abruptly

Description	Stops all axes moving immediately, ignoring any acceleration and jerk settings for each axis. Positional accuracy may be lost and re-initialisation of individual axes is recommended.
Command	`controller.stop.abruptly`
Parameters	None
Result	“0”

3.1.6. controller.serialnumber.get

Description	Returns controller serial number.
Command	`controller.serialnumber.get`
Parameters	None
Result	E.g. “577892”

3.1.7. controller.flag.get

Description	Returns a generic flag as an unsigned 32-bit value as hex string. The flag value is “0” following a power on. The user is free to use as required. A common use is to have it as a warm start flag, whereby after Connect() you can determine whether the controller has been powered off since last disconnect
Command	`controller.flag.get`
Parameters	None
Result	32-bit Flag value in HEX format ie ABCD1234

3.1.8. controller.flag.set

Description	Sets a generic flag as an unsigned 32-bit integer.		
Command	`controller.flag.set <f>`		
Parameters	`<f>`	32-bit value as hex string	*string*
Result	“0”		

3.2. Stage Commands

3.2.1. controller.stage.busy.get

Description	Gets the busy (moving) status of the stage
Command	`controller.stage.busy.get`
Parameters	None
Result	"0" idle, "1" X moving, "2" Y moving, "3" both X&Y moving

3.2.2. controller.stage.position.get

Description	Returns the current stage XY position By default, units for stage position are integer representation of microns. If sub-micron resolution is required and the stage/controller supports it then the user units can be changed. See *controller.stage.ss.set*
Command	`controller.stage.position.get`
Parameters	None
Result	"X,Y" ie "1234,5678"

3.2.3. controller.stage.position.set

Description	Sets the current physical position to the specified position is current user units. Positions can only be set when stage is not busy.
Command	`controller.stage.position.set <X> <Y>`
Parameters	`<X>` | New X position | *int* `<Y>` | New Y position | *int*
Result	"0"

3.2.4. controller.stage.goto-position

Description	Request the stage to move to the given position using the existing speed, acceleration and curve settings. The controller will attempt to change these parameters for the axis moving the shortest distance in order to synchronise the end of movements but it does not guarantee this.
Command	`controller.stage.goto-position <X> <Y>`
Parameters	`<X>` | X-target position | *int* `<Y>` | Y- target position | *int*
Result	"0"

3.2.5. controller.stage.move-at-velocity

Description	Request the stage to move at a constant velocity of X and Y microns/s. This is a float value and the controller will round that down to the next whole microstep velocity.
Command	`controller.stage.move-at-velocity <X> <Y>`
Parameters	`<X>` X-velocity *float* `<Y>` Y-velocity *float*
Result	“0”

3.2.6. controller.stage.name.get

Description	Return the name of the stage attached.
Command	`controller.stage.name.get`
Parameters	none
Result	“H101/A” for instance, or “NONE” if no stage

3.2.7. controller.stage.steps-per-micron.get

Description	Returns the number of whole microsteps per micron.
Command	`controller.stage.steps-per-micron.get`
Parameters	None
Result	“25” for instance. This number varies depending on the stage motor/lead screw combination for stepper motor stages or encoder resolution on linear stages. For this example setting *controller.stage.ss.set* to 1 gives a user unit of 0.04microns

3.2.8. controller.stage.limits.get

Description	Returns the limit switch state for the XY axes of the controller
Command	`controller.stage.limits.get`
Parameters	None
Result	An integer representing an 4 bit unsigned value with the following bit usage Bit: 3 = Y-, 2 = Y+, 1 = X-, 0 = X+ A 1 in a bit position indicates logical switch active state.

Bit	3	2	1	0
switch	Y-	Y+	X-	X+

3.2.9. controller.stage.speed.get

Description	Returns the maximum speed during a point to point move
Command	`controller.stage.speed.get`
Parameters	None
Result	An integer representing the speed in microns/s

3.2.10. controller.stage.speed.set

Description	Sets the maximum speed during a point to point move
Command	`controller.stage.speed.set <max speed>`
Parameters	`<max speed>` | Max speed in microns/s | *int*
Result	“0”

3.2.11. controller.stage.acc.get

Description	Gets the maximum acceleration during a point to point move or velocity move
Command	`controller.stage.acceleration.get`
Parameters	None
Result	An integer representing the acceleration in microns/s/s

3.2.12. controller.stage.acc.set

Description	Sets the maximum acceleration during a point to point move or velocity move
Command	`controller.stage.acceleration.set <maxacc>`
Parameters	`<maxacc>` | Max acceleration in microns/s/s | *int*
Result	“0”

3.2.13. controller.stage.jerk.get

Description	Gets the jerk time during a point to point move
Command	`controller.stage.jerk.get`
Parameters	None
Result	An integer representing the time in milliseconds before constant acceleration phase.

3.2.14. controller.stage.jerk.set

Description	Sets the jerk time during a point to point move
Command	`controller.stage.jerk.set <time>`
Parameters	`<time>` | Jerk time in milliseconds | *int*
Result	“0”

3.2.15. controller.stage.hostdirection.set

Description	Sets the physical direction each stage axis will move given an increasing +ve position. By default positively increasing XY positions will move the stage to its front right position
Command	`controller.stage.hostdirection.set <X> <Y>`
Parameters	`<X>` Direction [-1,1] *int* `<Y>` Direction [-1,1] *int*
Result	"0"

3.2.16. controller.stage.joystickdirection.set

Description	Sets the physical direction each axis will move in relation to the joystick deflection. By default when looking at the top plate of the stage it will seem to move in the same direction as the deflection of the joystick.
Command	`controller.stage.joystickdirection.set <X> <Y>`
Parameters	`<X>` Direction [-1,1] *int* `<Y>` Direction [-1,1] *int*
Result	"0"

3.2.17. controller.stage.joyxyz.on

Description	Enables the joystick.
Command	`controller.stage.joyxyz.on`
Parameters	None
Result	"0"

3.2.18. controller.stage.joyxyz.off

Description	Disables the joystick.
Command	`controller.stage.joyxyz.off`
Parameters	None
Result	"0"

3.2.19. controller.stage.ss.get

Description	Gets the current user unit step size. By default, the DLL works in user units of whole microns. This value represents the number of micro-steps per micron. This value varies depending on motor type and stage construction. For example, a H101A stage has a 200-step motor and 2mm pitch lead screw. Prior controllers micro-step at 250 steps/full step therefore there are 50000 micro-steps/rev of the motor. 2mm / 50000 = 0.04microns. So theoretically setting SS to 1 results in user unit of 0.04microns, or multiples thereof. In practice, this may not be physically possible due to motor behaviour and mechanical limitations. See also *controller.stage.steps-per-micron.get*
Command	`controller.stage.ss.get`
Parameters	None
Result	Typical responses for a stepper stage are "25", "50" and "100". For a linear stage fitted with typical 50nm encoders the response will be "20"

3.2.20. controller.stage.ss.set

Description	Sets the current user unit step size.
Command	`controller.stage.ss.set <ss>`
Parameters	`<ss>` Micro-steps per user unit *int*
Result	"0"

3.2.21. controller.stage.backlash.get

Description	gets the electronic stage backlash parameters
Command	`controller.stage.backlash.get`
Parameters	None
Result	"e,b" where e = enabled [0|1], b = backlash correction in microns. EG "1,10"

3.2.22. controller.stage.backlash.set

Description	sets the electronic stage backlash parameters
Command	`controller.stage.backlash.set <e> <b>`
Parameters	`<e>` Enabled [0|1] *int*; `<b>` Backlash distance in microns *int*
Result	"0"

3.3. Z (focus) Commands

3.3.1. controller.z.busy.get

Description	Gets the busy (moving) status of the Z (focus) axis
Command	`controller.z.busy.get`
Parameters	None
Result	“0” idle, “4” Z moving

3.3.2. controller.z.name.get

Description	Return the name of the Z (focus) device attached.
Command	`controller.z.name.get`
Parameters	none
Result	“OPENSTAND” or “NORMAL” for instance, or “NONE” if no stage

3.3.3. controller.z.limits.get

Description	Returns the limit switch state for the Z axis of the controller
Command	`controller.z.limits.get`
Parameters	None
Result	An integer representing an 2 bit unsigned value with the following bit usage Bit: 1 = Z-, 0 = Z+ A 1 in a bit position indicates logical switch active state.

Bit	1	0
switch	Z-	Z+

3.3.4. controller.z.microns-per-rev.get

Description	Returns the number of whole microns of focus movement that one revolution of the motor causes. The default value is 100, which is typical for a fine focus of a microscope. Other ‘known’ Prior focus devices will automatically set their values.
Command	`controller.z.microns-per-rev.get`
Parameters	None
Result	“100” for instance.

3.3.5. controller.z.microns-per-rev.set

Description	Sets the number of whole microns of focus movement that one revolution of the motor causes. The default value of 100 is a typical value for a fine focus of a microscope.
Command	`controller.z.microns-per-rev.set <upr>`
Parameters	`<upr>` | Microns per revolution of the fine focus mechanism | *int*
Result	"0"

3.3.6. controller.z.position.get

Description	Returns the current stage Z position By default, units for stage position are integer representation of 100nm steps sizes. See *controller.z.ss.set*
Command	`controller.z.position.get`
Parameters	None
Result	ie "12345" interpreted as 1234.5 microns if default used

3.3.7. controller.z.position.set

Description	Sets the current physical position to the specified position is current user units. Positions can only be set when Z is not busy.
Command	`controller.z.position.set <Z>`
Parameters	`<Z>` | New Z position | *int*
Result	"0"

3.3.8. controller.z.goto-position

Description	Request the Z to move to the given position using the existing speed, acceleration and curve settings.
Command	`controller.z.goto-position <Z>`
Parameters	`<Z>` | Z-target position | *int*
Result	"0"

3.3.9. controller.z.move-at-velocity

Description	Request the Z to move at a constant velocity of Z microns/s. This is a float value and the controller will round that down to the next whole micro-step velocity.
Command	`controller.z.move-at-velocity <Z>`
Parameters	`<Z>` | Z-velocity | *float*
Result	"0"

3.3.10. controller.z.speed.get

Description	Returns the maximum speed during a point to point move
Command	`controller.z.speed.get`
Parameters	None
Result	An integer representing the speed in microns/s

3.3.11. controller.z.speed.set

Description	Sets the maximum speed during a point to point move
Command	`controller.z.speed.set <max speed>`
Parameters	`<max speed>` | Max speed in microns/s | *int*
Result	“0”

3.3.12. controller.z.acc.get

Description	Gets the maximum acceleration during a point to point move or velocity move
Command	`controller.z.acceleration.get`
Parameters	None
Result	An integer representing the acceleration in microns/s/s

3.3.13. controller.z.acc.set

Description	Sets the maximum acceleration during a point to point move or velocity move
Command	`controller.z.acceleration.set <maxacc>`
Parameters	`<maxacc>` | Max acceleration in microns/s/s | *int*
Result	“0”

3.3.14. controller.z.jerk.get

Description	Gets the jerk time during a point to point move
Command	`controller.z.jerk.get`
Parameters	None
Result	An integer representing the time in milliseconds before constant acceleration phase.

3.3.15. controller.z.jerk.set

Description	Sets the jerk time during a point to point move
Command	`controller.z.jerk.set <time>`
Parameters	`<time>` | Jerk time in milliseconds | *int*
Result	"0"

3.3.16. controller.z.hostdirection.set

Description	Sets the physical direction the z axis will move given an increasing +ve position.
Command	`controller.z.hostdirection.set <Z>`
Parameters	`<Z>` | Direction [-1,1] | *int*
Result	"0"

3.3.17. controller.z.joystickdirection.set

Description	Sets the physical direction each axis will move in relation to the z digipot rotation.
Command	`controller.z.joystickdirection.set <Z>`
Parameters	`<>` | Direction [-1,1] | *int*
Result	"0"

3.3.18. controller.z.ss.get

Description	Gets the current user unit step size for Z. By default, the DLL works in user units of 100nm. This value represents the number of micro-steps per 100nm. This value varies depending on motor type and microns-per–rev setting. For example, a NORMAL focus motor on a microscope with 100microns fine focus has 50000 micro-steps/rev of the motor. 100um / 50000 = 2nm. So theoretically setting SS to 1 results in user unit of 2nm, or multiples thereof. In practice, this may not be physically possible due to motor behaviour and mechanical limitations. See also controller.z.microns-per-rev.get
Command	`controller.z.ss.get`
Parameters	None
Result	Typical responses are : "5" for FB20X at 1000 microns/rev and "50" for NORMAL motor attached to focus knob of microscope with 100um per revolution of the fine focus.

3.3.19. controller.z.ss.set

Description	Sets the current user unit step size.
Command	`controller.z.ss.set <ss>`
Parameters	`<ss>` | Micro-steps per user unit | *int*
Result	"0"

3.3.20. controller.z.backlash.get

Description	gets the electronic z (focus) backlash parameters
Command	`controller.z.backlash.get`
Parameters	None
Result	"e,b" where e = enabled [0|1], b = backlash correction in microns. EG "1,10"

3.3.21. controller.z.backlash.set

Description	sets the electronic z (focus) backlash parameters		
Command	`controller.z.backlash.set <e> <b>`		
Parameters	`<e>`	Enabled [0|1]	*int*
	`<b>`	Backlash distance in microns	*int*
Result	"0"		

3.4. Filter Commands

3.4.1. controller.filter.fitted.get

Description	Gets the fitted status of the specified filter wheel.
Command	`controller.filter.fitted.get <f>`
Parameters	`<f>` Filter Id [1..6] *int*
Result	“0” not fitted “1” fitted

3.4.2. controller.filter.name.get

Description	Gets the name of the specified filter wheel.
Command	`controller.filter.name.get <f>`
Parameters	`<f>` Filter Id [1..6] *int*
Result	Eg “HF108-8”

3.4.3. controller.filter.filters-per-wheel.get

Description	Gets the number of filters on the wheel.
Command	`controller.filter.filter-per-wheel.get <f>`
Parameters	`<f>` Filter Id [1..6] *int*
Result	Eg “8”

3.4.4. controller.filter.position.get

Description	Gets the current filter wheel position.
Command	`controller.filter.position.get <f>`
Parameters	`<f>` Filter Id [1..6] *int*
Result	Eg “8”

3.4.5. controller.filter.goto-position

Description	Move to requested filter position.
Command	`controller.filter.goto-position <f> <p>`
Parameters	`<f>` Filter Id [1..6] *int* `<p>` Filter pos [1..filter-per-wheel] *int*
Result	“0”

3.4.6. controller.filter.home

Description	Homes the specified wheel. Wheel will spin around, finding its alignment and finish in position 1.
Command	`controller.filter.home <f>`
Parameters	`<f>` Filter Id [1..6] *int*
Result	“0”

3.4.7. controller.filter.busy.get

Description	Gets the busy (moving) status
Command	`controller.filter.busy.get <f>`
Parameters	`<f>` Filter Id [1..6] *int*
Result	“0” idle “1” busy

3.4.8. controller.filter.speed.get

Description	Get the speed in percentage terms of the recommended Prior default value.
Command	`controller.filter.speed.get <f>`
Parameters	`<f>` Filter Id [1..6] *int*
Result	“0”.. “100”

3.4.9. controller.filter.speed.set

Description	Adjusts the speed in percentage terms of the recommended Prior default value. Although the speed can be increased above 100% there is the possibility that motor will stall and lose positional accuracy.
Command	`controller.filter.speed.set <f> <s>`
Parameters	`<f>` Filter Id [1..6] *int* `<s>` Percentage of recommended speed *int*
Result	“0”

3.4.10. controller.filter.acc.get

Description	Get the acceleration in percentage terms of the recommended Prior default value.
Command	`controller.filter.acc.get <f>`
Parameters	`<f>` Filter Id [1..6] *int*
Result	“0”.. “100”

3.4.11. controller.filter.acc.set

Description	Adjusts the acceleration in percentage terms of the recommended Prior default value. Although the acceleration can be increased above 100% there is the possibility that motor will stall and lose positional accuracy.
Command	`controller.filter.speed.set <f> <s>`
Parameters	`<f>` Filter Id [1..6] *int* `<s>` Percentage of recommended acceleration *int*
Result	“0”

3.4.12. controller.filter.jerk.get

Description	Gets the jerk time period for any move
Command	`controller.filter.jerk.get <f>`
Parameters	`<f>` Filter Id [1..6] *int*
Result	An integer representing the time in milliseconds before constant acceleration phase.

3.4.13. controller.filter.jerk.set

Description	Sets the jerk time during any move
Command	`controller.filter.jerk.get <f> <time>`
Parameters	`<f>` Filter Id [1..6] *int* `<time>` Jerk time in milliseconds *int*
Result	An integer representing the time in milliseconds before constant acceleration phase.

3.5. Shutter Commands

3.5.1. controller.shutter.fitted.get

Description	Gets the fitted status of the specified shutter.
Command	`controller.shutter.fitted.get <s>`
Parameters	`<s>` Shutter Id [1..6] *int*
Result	“0” not fitted “1” fitted

3.5.2. controller.shutter.name.get

Description	Gets the name of the specified shutter
Command	`controller.shutter.name.get <s>`
Parameters	`<s>` Shutter Id [1..6] *int*
Result	Eg “NORMAL”

3.5.3. controller.shutter.open

Description	Open the specified shutter
Command	`controller.shutter.open <s>`
Parameters	`<s>` Shutter Id [1..6] *int*
Result	0

3.5.4. controller.shutter.close

Description	Close the specified shutter
Command	`controller.shutter.close <s>`
Parameters	`<s>` Shutter Id [1..6] *int*
Result	0

3.6. Trigger Commands

3.6.1. controller.trigger.resolution.get

Description	Returns the number of encoder counts per micron for the given XYZ axis. All triggers points are specified in terms of raw encoder count. The user application stage positions must be converted from local user units into encoder counts.
Command	`controller.trigger.resolution.get <axis>`
Parameters	`<axis>` | Axis 'X','Y', or 'Z' | *char*
Result	"0"

3.6.2. controller.trigger.arm

Description	Create and arm a trigger sequence. Example: A single chord in X with first trigger at 0, followed by 19 triggers every +100 counts, -ve trigger pulse of 1 ms (millisecond) duration. Step 1: Position the stage **before** the first intended trigger point. Step 2: Send 'TRIGGER 0,100,X,20,N,1000'to arm trigger mechanism. Step 3: Move stage over intended triggers (any command or even joystick movement). The triggers will output when X = 0, 100, 200....1800, 1900 encoder counts. The trigger mechanism is automatically disarmed after all specified triggers have been output. Negating the sign of D can be used to trigger in reverse direction. The user application should convert local stage position to encoder counts when using the TRIGGER function. By default the PS3 XY position is reported in microns, the Z position is 100nm steps
Command	`controller.trigger.arm <F> <D> <A> <N> <P> <W>`
Parameters	`<F>` | First trigger position in encoder counts | *int* `<D>` | Distance between triggers in encoder counts | *int* `<A>` | Axis to trigger from 'X', 'Y' or 'Z' | *char* `<N>` | Number of triggers in chord | *int* `<P>` | trigger pulse polarity 'P' or 'N' | *char* `<W>` | trigger pulse in microseconds | *int*
Result	"0"

3.7. TTL Commands

3.7.1. controller.ttl.in.get

Description	Returns the current tll input state
Command	`controller.ttl.in.get`
Parameters	None
Result	Integer representing the binary pin state of available TTL inputs TTLIN3..0

3.7.2. controller.ttl.out.get

Description	Returns the current tll output state
Command	`controller.ttl.in.get`
Parameters	None
Result	Integer representing the binary pin state of available TTL outputs TTLOUT3..0

3.7.3. controller.ttl.out.set

Description	Returns the current tll input state		
Command	`controller.ttl.out.set <state>`		
Parameters	`<state>`	Decimal 0..15 for binary output pins TTLOUT3..0	*int*
Result	"0"		

3.8. Led Commands

3.8.1. controller.led.fitted.get

Description	Gets the fitted status of the specified led.
Command	`controller.led.fitted.get <l>`
Parameters	`<l>` | led Id [1..8] | *int*
Result	“0” not fitted “1” fitted

3.8.2. controller.led.power.get

Description	Gets the power level of the specified led in percent
Command	`controller.led.power.get <l>`
Parameters	`<l>` | led Id [1..8] | *int*
Result	“0” .. “100”

3.8.3. controller.led.power.set

Description	Sets the power level of the specified led in percent.
Command	`controller.led.power.set <l> <p>`
Parameters	`<l>` | led Id [1..8] | *int* `<p>` | 0..100 | *int*
Result	“0”

3.8.4. controller.led.state.get

Description	Gets the state of the specified led.
Command	`controller.led.state.get <l>`
Parameters	`<l>` | led Id [1..8] | *int*
Result	“0” off, “1” on

3.8.5. controller.led.state.set

Description	Sets the state of the specified led.
Command	`controller.led.state.set <l> <s>`
Parameters	`<l>` | led Id [1..8] | *int* `<s>` | 0..1 | *int*
Result	“0”

3.8.6. controller.led.fan.get

Description	Sets the on/off state of the specified fan.		
Command	`controller.led.fan.get <f>`		
Parameters	`<l>`	led Id [1..8]	*int*
Result	“0” off, “1” on		

3.8.7. controller.led.fan.set

Description	Sets the on/off state of the specified fan.		
Command	`controller.led.fan.set <l> <s>`		
Parameters	`<l>`	led Id [1..8]	*int*
	`<s>`	0..1	*int*
Result	“0”		

3.8.8. controller.led.fluor.get

Description	Gets the fluor description of the led.		
Command	`controller.led.fluor.get <l>`		
Parameters	`<l>`	led Id [1..8]	*int*
Result	Eg “TRITC”		

3.8.9. controller.led.lambda.get

Description	Gets the wavelength of the led.		
Command	`controller.led.lambda.get <l>`		
Parameters	`<l>`	led Id [1..8]	*int*
Result	Eg “525”		

3.8.10. controller.led.temperature.get

Description	Gets the temperature (degrees) of the led.		
Command	`controller.led.temperature.get <l>`		
Parameters	`<l>`	led Id [1..8]	*int*
Result	Eg “30”		

3.9. OEM Commands

OEM axes are stepper motors fitted to the filter axes (1..6) that identify as having normally open or normally closed limit switches depending on their plug and play identifiers. This allows the user to drive them directly as they wish to create OEM applications. Their positions, speeds and accelerations are in units of microsteps or if fitted, encoder resolution.

3.9.1. controller.oem.config

Description	For an axis that has no plug and play identifier the user should configure the axis function. These functions are pre-defined and require the appropriate Prior devices to be fitted (contact Prior for advice). An example of this would be a 2 or 6 position objective changer. For these devices positions are logical device positions, ie an objective identifier. Example: “controller.oem.config 1 HH339” configures filter 1 drive as a 6 position nosepiece.
Command	`controller.oem.config <id> <name>`
Parameters	`<id>` · oem Id [1..6] · *int* `<name>` · Device id · *string*
Result	“0”

3.9.2. controller.oem.position.get

Description	Get the current position of the oem axis. Value returned is in either microsteps, or encoder counts or a logical device position depending on configuration.
Command	`controller.oem.position.get <id>`
Parameters	`<id>` · oem Id [1..6] · *int*
Result	Device position

3.9.3. controller.oem.position.set

Description	Set the current position of the oem axis. Value returned is in either microsteps, or encoder counts or a logical device position depending on configuration. Pre-configured devices may not allow their position to be modified.
Command	`controller.oem.position.set <id> <p>`
Parameters	`<id>` · oem Id [1..6] · *int* `<p>` · position · *int*
Result	“0”

3.9.4. controller.oem.goto-position

Description	Drive the oem axis to the specified position. Units will be microsteps, encoder counts or logical positions depending on configuration.		
Command	`controller.oem.goto-position <id> <pos>`		
Parameters	`<id>`	oem Id [1..6]	*int*
	`<pos>`	Target position	*int*
Result	“0”		

3.9.5. controller.oem.move-at-velocity

Description	Drive the oem axis at the specified velocity. Units will be microsteps, encoder counts or logical positions depending on configuration. Pre-configured devices may not allow velocity movements.		
Command	`controller.oem.move-at-velocity <id> <vel>`		
Parameters	`<id>`	oem Id [1..6]	*int*
	`<vel>`	Target velocity	*int*
Result	“0”		

3.9.6. controller.oem.busy.get

Description	Determine whether the oem axis is busy (ie moving)		
Command	`controller.oem.busy.get <id>`		
Parameters	`<id>`	oem Id [1..6]	*int*
Result	“0” idle, “1” moving		

3.9.7. controller.oem.speed.get

Description	Get the maximum speed of the axis used during a move. This will be in microsteps/s or encoder counts/s.		
Command	`controller.oem.speed.get <id>`		
Parameters	`<id>`	oem Id [1..6]	*int*
Result	Eg “600000”		

3.9.8. controller.oem.speed.set

Description	Set the maximum speed of the axis used during a move. This will be in microsteps/s or encoder counts/s. Pre-defined devices may not allow speed adjustment.		
Command	`controller.oem.speed.set <id> <s>`		
Parameters	`<id>`	oem Id [1..6]	*int*
	`<s>`	Max speed	*int*
Result	“0”		

3.9.9. controller.oem.acc.get

Description	Get the maximum acceleration of the axis used during a move. This will be in microsteps/s/s or encoder counts/s/s.		
Command	`controller.oem.acc.get <id>`		
Parameters	`<id>`	oem Id [1..6]	*int*
Result	Eg “2855000”		

3.9.10. controller.oem.acc.set

Description	Set the maximum acceleration of the axis used during a move. This will be in microsteps/s/s or encoder counts/s/s. Pre-defined devices may not allow acc adjustment.		
Command	`controller.oem.acc.set <id> <a>`		
Parameters	`<id>`	oem Id [1..6]	*int*
	`<a>`	Max acc	*int*
Result	“0”		

3.9.11. controller.oem.jerk.get

Description	Get the jerk of the axis used during a move. This will be in milliseconds		
Command	`controller.oem.jerk.get <id>`		
Parameters	`<id>`	oem Id [1..6]	*int*
Result	Eg “13” ms		

3.9.12. controller.oem.jerk.set

Description	Get the jerk of the axis used during a move. This will be in milliseconds. Pre-defined devices may not allow speed adjustment.		
Command	`controller.oem.jerk.get <id> <j>`		
Parameters	`<id>`	oem Id [1..6]	*int*
	`<j>`	Jerk (ms)	*int*
Result	“0”		

3.9.13. controller.oem.limits.get

Description	Get the active limits switch status of the axis. Pre-defined devices may not allow speed adjustment.		
Command	`controller.oem.limits.get <id>`		
Parameters	`<id>`	oem Id [1..6]	*int*
Result	“0” = no limits active, “1” = +ve switch active, “2” = -ve switch active		

3.9.14. controller.oem.home

Description	Home the axis. For a normal device with limit switches, this will move the axis to the –ve limit switch.
Command	`controller.oem.limits.get <id>`
Parameters	`<id>` | oem Id [1..6] | *int*
Result	“0” = no limits active, “1” = +ve switch active, “2” = -ve switch active

4. ODS Loader Commands

4.1. ods.connect

Description	Establish a communications connection between the DLL and the ODS loader on the specified port.
Command	`ods.connect <port>`
Parameters	`<port>` This is numerical number of the communications port listed in the device manager under 'Ports (COM & LPT)'
Result	"0"

4.2. ods.disconnect

Description	Closes the currently open communications channel to the ODS loader
Command	`ods.disconnect`
Parameters	None
Result	"0"

4.3. ods.status.get

Description	Get the status word from the ODS loader. See *ODS Status Word* for bit values.
Command	`ods.status.get`
Parameters	None
Result	Decimal integer corresponding to bits in the Status Word

4.4. ods.initialise

Description	After Connect to the loader has occurred, it will be in an un-initialised state and must first be initialised before any other action can be performed. From cold power on condition, the loader will move all axes to known datum points to establish its reference positions. If the loader had not been powered off during its last use, then establishing reference points is not needed and the routine returns immediately.
Command	`ods.initialise`
Parameters	None
Result	Decimal integer corresponding to bits in the Status Word

4.5. ods.scanhotel

Description	When a hotel is fitted and detected (see *ods.hotelfitted.get*) it must first be scanned in order to detect which apartments have plates fitted. After scanning, the plates fitted can be determined via *ods.platefitted.get.*		
Command	`ods.scanhotel <hotel>`		
Parameters	`<hotel>`	Hotel id [1|2]	*int*
Result	"0"		

4.6. ods.movetostage

Description	Request to move a plate from a hotel apartment to the stage		
Command	`ods.movetostage <hotel> <apartment>`		
Parameters	`<hotel>`	Hotel id [1..*ods.maxhotels.get*]	*int*
	`<apartment>`	Apartment id [1..*ods.maxplatesperhotel.get*]	*int*
Result	"0"		

4.7. ods.movetohotel

Description	Request to move a plate from the stage to a hotel apartment		
Command	`ods.movefromstage <hotel> <apartment>`		
Parameters	`<hotel>`	Hotel id [1..*ods.maxhotels.get*]	*int*
	`<apartment>`	Apartment id [1..*ods.maxplatesperhotel.get*]	*int*
Result	"0"		

4.8. ods.stop

Description	Stop the loader and return to the idle state
Command	`ods.stop`
Parameters	None
Result	"0"

4.9. ods.lasterror.get

Description	If the DLL API returns a PRIOR_LOADERERROR then the reason can be determined via this call. Similarly if the ODS_LOADER_ERROR error bit is set in the status word during a loader function (ie move to stage)
Command	`ods.lasterror.get`
Parameters	None
Result	Decimal string see *ODS Get Last Error codes*

4.10. ods.lasterror.clear

Description	Clears the last loader error flag to zero.
Command	`ods.lasterror.clear`
Parameters	None
Result	“0”

4.11. ods.stalledaxis.get

Description	If the ODS_LOADER_AXISSTALLED bit is set in the *ODS Status Word* then this returns the offending axis id.
Command	`ods.stalledaxis.get`
Parameters	None
Result	“0”

4.12. ods.hotelfitted.get

Description	Determine what hotels are fitted
Command	`ods.hotelfitted.get <hotel>`
Parameters	`<hotel>` | Hotel id [1..*ods.maxhotels.get*] | *int*
Result	“0” not fitted, or “1” fitted

4.13. ods.platefitted.get

Description	Determine what plates are fitted
Command	`ods.platefitted.get <hotel> <apartment>`
Parameters	`<hotel>` | Hotel id [1..*ods.maxhotels.get*] | *int* `<apartment>` | Apartment id [1..*ods.maxplatesperhotel.get*] | *int*
Result	“0” not fitted, or “1” fitted

4.14. ods.maxhotels.get

Description	Determine the maximum number of supported hotels
Command	`ods.maxhotels.get`
Parameters	None
Result	Decimal string representing max hotels count

4.15. ods.maxplatesperhotel.get

Description	Determine the maximum number of apartments (plates) in hotel
Command	`ods.maxplatesperhotel.get`
Parameters	None
Result	Decimal string representing max hotel apartments (plates)

4.16. ods.axis.jog

Description	Used during initialisation and initial setup/calibration to manually jog the loader axis relative to its current position.		
Command	`ods.axis.jog <axis> <distance>`		
Parameters	`<axis>`	See *ODS Loader Axes*	*int*
	`<distance>`	Encoder counts	*int*
Result	“0”		

4.17. ods.axis.goto

Description	Used during setup/calibration to manually move the loader to a known absolute position. Do not use during initialisation as until initialisation has completed absolute positions are not valid.		
Command	`ods.axis.goto <axis> <absolute position>`		
Parameters	`<axis>`	See *ODS Loader Axes*	*int*
	`<absolute position>`	Encoder counts	*int*
Result	“0”		

4.18. ods.axis.busy.get

Description	Used to determine whether axis is currently moving. Only needed during setup/calibration when manually moving the loader.		
Command	`ods.axis.busy.get <axis>`		
Parameters	`<axis>`	See *ODS Loader Axes*	*int*
Result	“0” axis idle, “1” axis busy		

4.19. ods.sethotelposition

Description	Stores the current position of the loader as the calibrated hotel position. Only needed during initial stage calibration.		
Command	`ods.sethotelposition <hotel>`		
Parameters	`<hotel>`	Hotel id [1..*ods.maxhotels.get*]	*int*
Result	“0”		

4.20. ods.setstageposition

Description	Stores the current position of the loader as the calibrated stage position. Only needed during initial stage calibration.
Command	`ods.setstageposition`
Parameters	None
Result	“0”

4.21. ods.setupcomplete

Description	Called at the end of the calibration process to save the setup data for the loader to disk.
Command	`ods.setupcomplete`
Parameters	None
Result	“0”

4.22. ods.reloadsetup

Description	Reload the setup with immediate effect on loader positions. Useful during initial calibration when manually tweaking calibrated hotel and stage positions.
Command	`ods.reloadsetup`
Parameters	None
Result	“0”

4.23. ods.singlestepmode.set

Description	Activate the single step mode of the loader. This is a useful debug facility for stepping through the loaders actions
Command	`ods.singlestepmode.set <mode>`
Parameters	`<mode>` | “0” off, “1” on | *int*
Result	“0”

4.24. ods.singlestep

Description	With single step mode activated, this command causes the loader to move one-step through its current action state machine. This is a useful debug facility for stepping through the loaders actions such as transferring plates etc
Command	`ods.singlestep`
Parameters	None
Result	“0”

4.25. ods.firmwareversion

Description	Return the firmware version of the loaders axis controllers.
Command	`ods.firmwareversion.get <axis>`
Parameters	`<axis>` | See *ODS Loader Axes* | *int*
Result	Firmware version string ie “0.23”

4.26. ods.transferflag.set

Description	This flag is used during the setup process only to modify the behaviour of the plate transfer process when setting the stage plate loading position
Command	`ods.transferflag.set <value>`
Parameters	`<value>` | Currently “1” and is self cancelling | *int*
Result	“0”

4.27. ods.serialnumber.get

Description	Return the serial number of the loader
Command	`ods.serialnumber.get`
Parameters	None
Result	Serial number of the loader

4.28. ods.serialnumber.set

Description	Set the serial number during the setup/calibration process. This value is stored in the INI calibration file.
Command	`ods.serialnumber.set <serial>`
Parameters	`<serial>` | Serial number | *int*
Result	“0”

5. SL160 Loader Commands

5.1. sl160.connect

Description	Establish a communications connection between the DLL and the SL160 loader on the specified port. NOTE: connection to stage controller must be done first.
Command	`sl160.connect <port>`
Parameters	`<port>` — This should be the same port number as used when establishing the connection to the stage controller. The standard ProScan3 controller controls SL160 functions. — *int*
Result	“0”

5.2. sl160.disconnect

Description	Closes the currently open communications channel to the SL160 loader
Command	`sl160.disconnect`
Parameters	None
Result	“0”

5.3. sl160.status.get

Description	Get the status word from the SL160 loader. See *SL160 Status Word* for bit values.
Command	`sl160.status.get`
Parameters	None
Result	Decimal integer corresponding to bits in the Status Word

5.4. sl160.initialise

Description	After Connect to the loader has occurred, it enters the un-initialised state. From cold power on condition, the loader will move all axes to known datum points to establish its reference positions. If the loader had been left powered on following its last use, then establishing reference points is not needed and the routine returns immediately.
Command	`sl160.initialise`
Parameters	None
Result	“0”

5.5. sl160.scanhotel

Description	When a hotel is fitted and detected (see *sl160.hotelfitted.get*) it must first be scanned in order to detect which apartments have plates fitted. After scanning, the plates fitted can be determined via *sl160.trayfitted.get*
Command	`sl160.scanhotel <hotel>`
Parameters	`<hotel>` Hotel id [1..sl160.maxhotels.get] *int*
Result	“0”

5.6. sl160.movetostage

Description	Request to move a tray from a hotel apartment to the stage
Command	`sl160.movetostage <hotel> <apartment>`
Parameters	`<hotel>` Hotel id [1..sl160.maxhotels.get] *int* `<apartment>` Apartment id [1..sl160.maxtraysperhotel.get] *int*
Result	“0”

5.7. sl160.movetohotel

Description	Request to move a tray from the stage to a hotel apartment
Command	`sl160.movefromstage <hotel> <apartment>`
Parameters	`<hotel>` Hotel id [1..sl160.maxhotels.get] *int* `<apartment>` Apartment id [1..sl160.maxtraysperhotel.get] *int*
Result	“0”

5.8. sl160.stop

Description	Stop the loader immediately and return to the idle state. May require some user intervention.
Command	`sl160.stop`
Parameters	None
Result	“0”

5.9. sl160.previewstate.get

Description	When transferring a tray from hotel to the stage the loader will pause at preview stations, allowing an external preview camera to take an image of slides 1,2,3 & 4.
Command	`sl160.previewstate.get`
Parameters	None
Result	“0” - not at a preview station “n” - waiting at preview ‘n’ station

5.10. sl160.previewstate.set

Description	Cancel the preview state after preview image taken. Causes loader to move to next preview point or continue to load to stage
Command	`sl160.previewstate.set <state>`
Parameters	`<state>` | 0 | *int*
Result	“0”

5.11. sl160.unloadhotels

Description	Causes the loader to position hotels to the unload position so user can replace them.
Command	`sl160.unloadhotels`
Parameters	None
Result	“0”

5.12. sl160.loadhotels

Description	Determine what hotels user has placed on the shuttle and loads them ready for scanning.
Command	`sl160.loadhotels`
Parameters	None
Result	“0”

5.13. sl160.lasterror.get

Description	If the DLL API returns a PRIOR_LOADERERROR then the reason can be determined via this call. Similarly if the SL160_LOADER_ERROR error bit is set in the status word during a loader function (ie move to stage)
Command	`sl160.lasterror.get`
Parameters	None
Result	Decimal string see *ODS Get Last Error codes*

5.14. sl160.lasterror.clear

Description	Clears the last loader error flag to zero.
Command	`sl160.lasterror.clear`
Parameters	None
Result	"0"

5.15. sl160.stalledaxis.get

Description	If the SL_LOADER_AXISSTALLED bit is set in the SL160 Status Word then this returns the offending axis id.
Command	`sl160.stalledaxis.get`
Parameters	None
Result	"0"

5.16. sl160.hotelfitted.get

Description	Determine what hotels are fitted		
Command	`sl160.hotelfitted.get <hotel>`		
Parameters	`<hotel>`	Hotel id [1..sl160.maxhotels.get]	*int*
Result	"0" not fitted, or "1" fitted		

5.17. sl160.trayfitted.get

Description	Determine what trays are fitted		
Command	`sl160.trayfitted.get <hotel> <apartment>`		
Parameters	`<hotel>`	Hotel id [1..*ods.maxhotels.get*]	*int*
	`<apartment>`	Apartment id [1..*ods.maxplatesperhotel.get*]	*int*
Result	"0" not fitted, or "1" fitted		

5.18. sl160.maxhotels.get

Description	Determine the maximum number of supported hotels
Command	`sl160.maxhotels.get`
Parameters	None
Result	Decimal string representing max hotels count

5.19. sl160.maxtraysperhotel.get

Description	Determine the maximum number of apartments (trays) in hotel
Command	`sl160.maxtraysperhotel.get`
Parameters	None
Result	Decimal string representing max hotel apartments (trays)

5.20. sl160.axis.jog

Description	Used during initialisation and initial setup/calibration to manually jog the loader axis relative to its current position.		
Command	`sl160.axis.jog <axis> <distance>`		
Parameters	`<axis>`	See *SL160 Loader Axes*	*int*
	`<distance>`	Encoder counts	*int*
Result	“0”		

5.21. sl160.axis.goto

Description	Used during setup/calibration to manually move the loader to a known absolute position. Do not use during initialisation as until initialisation has completed absolute positions are not valid.		
Command	`sl160.axis.goto <axis> <absolute position>`		
Parameters	`<axis>`	See *SL160 Loader Axes*	*int*
	`<absolute position>`	Encoder counts	*int*
Result	“0”		

5.22. sl160.axis.move-at-velocity

Description	Used during setup/calibration to manually move the loader at a given velocity.		
Command	`sl160.axis.move-at-velocity <axis> <velocity>`		
Parameters	`<axis>`	See *SL160 Loader Axes*	*int*
	`<velocity>`	Encoder counts/s	*int*
Result	“0”		

5.23. sl160.axis.busy.get

Description	Used to determine whether axis is currently moving. Only needed during setup/calibration when manually moving the loader.
Command	`sl160.axis.busy.get <axis>`
Parameters	`<axis>` See *SL160 Loader Axes* *int*
Result	"0" axis idle, "1" axis busy

5.24. sl160.axis.position.get

Description	Used to determine whether axis is currently moving. Only needed during setup/calibration when manually moving the loader.
Command	`sl160.axis.position.get <axis>`
Parameters	`<axis>` See *SL160 Loader Axes* *int*
Result	"0" axis idle, "1" axis busy

5.25. sl160.calibration.set

Description	Stores the current positions of the loader as the calibrated load/unload position. Only needed during initial stage calibration.
Command	`sl160.calibration.set`
Parameters	None
Result	"0"

5.26. sl160.calibration.save

Description	Saves the calibrated positions of the loader into the controller backup and creates a INI file in ProgramData/Prior folder. Only needed during initial stage calibration.
Command	`sl160.calibration.save`
Parameters	None
Result	"0"

5.27. sl160.calibration.stagexy.get

Description	Returns the calibrated stage XY position. The application must position the stage to this position before loading or unloading trays to the stage.
Command	`sl160.calibration.stagexy.get`
Parameters	None
Result	Calibrated stage position in microns from the stage back right limit switch ie "45087,23345"

5.28. sl160.reloadsetup

Description	Reload the setup with immediate effect on loader positions. Useful during initial calibration when manually tweaking calibrated hotel and stage positions.
Command	`sl160.reloadesetup`
Parameters	None
Result	“0”

5.29. sl160.singlestepmode.set

Description	Activate the single step mode of the loader. This is a useful debug facility for stepping through the loaders actions
Command	`sl160.singlestepmode.set <mode>`
Parameters	`<mode>` | “0” off, “1” on | *int*
Result	“0”

5.30. sl160.singlestep

Description	With single step mode activated, this command causes the loader to move one-step through its current action state machine. This is a useful debug facility for stepping through the loaders actions such as transferring trays etc
Command	`sl160.singlestep`
Parameters	None
Result	“0”

5.31. sl160.serialnumber.get

Description	Return the serial number of the loader
Command	`sl160.serialnumber.get`
Parameters	None
Result	Serial number of the loader

5.32. sl160.serialnumber.set

Description	Set the serial number during the setup/calibration process. This value is stored in the INI calibration file.
Command	`sl160.serialnumber.set <serial>`
Parameters	`<serial>` | Serial number | *int*
Result	“0”

6. WASLV2 Loader Commands

6.1. waslv2.connect

Description	Establish a communications connection between the DLL and the WASLV2 loader on the specified port. NOTE: connection to stage controller must be done first.
Command	`waslv2.connect <port>`
Parameters	`<port>` This should be the same port number as used when establishing the connection to the stage controller. The standard ProScan3 controller controls WASLV2 functions. *int*
Result	"0"

6.2. waslv2.disconnect

Description	Closes the currently open communications channel to the WASLV2 loader
Command	`waslv2.disconnect`
Parameters	None
Result	"0"

6.3. waslv2.status.get

Description	Get the status word from the WASLV2 loader. See *WASLV2 Status Word* for bit values.
Command	`waslv2.status.get`
Parameters	None
Result	Decimal integer corresponding to bits in the Status Word

6.4. waslv2.initialise

Description	After Connect to the loader has occurred, it enters the un-initialised state. From cold power on condition, the loader will move all axes to known datum points to establish its reference positions. If the loader had been left powered on following its last use, then establishing reference points is not needed and the routine returns immediately.
Command	`waslv2.initialise`
Parameters	None
Result	"0"

6.5. waslv2.scanhotel

Description	When a hotel is fitted and detected (see *waslv2.hotelfitted.get*) it must first be scanned in order to detect which apartments have plates fitted. After scanning, the plates fitted can be determined via *waslv2.trayfitted.get*
Command	`waslv2.scanhotel <hotel>`
Parameters	`<hotel>` — Hotel id [1..waslv2.maxhotels.get] — *int*
Result	“0”

6.6. waslv2.movetostage

Description	Request to move a tray from a hotel apartment to the stage
Command	`waslv2.movetostage <hotel> <apartment>`
Parameters	`<hotel>` — Hotel id [1..waslv2.maxhotels.get] — *int*
	`<apartment>` — Apartment id [1..waslv2.maxtraysperhotel.get] — *int*
Result	“0”

6.7. waslv2.movetohotel

Description	Request to move a tray from the stage to a hotel apartment
Command	`waslv2.movefromstage <hotel> <apartment>`
Parameters	`<hotel>` — Hotel id [1..waslv2.maxhotels.get] — *int*
	`<apartment>` — Apartment id [1..waslv2.maxtraysperhotel.get] — *int*
Result	“0”

6.8. waslv2.stop

Description	Stop the loader immediately and return to the idle state. May require some user intervention.
Command	`waslv2.stop`
Parameters	None
Result	“0”

6.9. waslv2.previewstate.get

Description	When transferring a tray from hotel to the stage the loader will pause at two preview stations, allowing an external preview camera to take an image of slides 1 & 2 when at preview point 1 and slides 3 & 4 when at preview point 2. Preview state should be polled, and user action taken when at position 1 or 2.
Command	`waslv2.previewstate.get`
Parameters	None
Result	"0" - not at a preview station "1" - waiting at preview 1 station "2" - waiting at preview 2 station

6.10. waslv2.previewstate.set

Description	Cancel the preview state after preview image taken. Causes loader to move to next preview point or continue to load to stage
Command	`waslv2.previewstate.set <state>`
Parameters	`<state>` | 0 | *int*
Result	"0"

6.11. waslv2.unloadhotels

Description	Causes the loader to position hotels to the unload position so user can replace them.
Command	`waslv2.unloadhotels`
Parameters	None
Result	"0"

6.12. waslv2.loadhotels

Description	Determine what hotels user has placed on the shuttle and loads them ready for scanning.
Command	`waslv2.loadhotels`
Parameters	None
Result	"0"

6.13. waslv2.lasterror.get

Description	If the DLL API returns a PRIOR_LOADERERROR then the reason can be determined via this call. Similarly if the WASLV2_LOADER_ERROR error bit is set in the status word during a loader function (ie move to stage)
Command	`waslv2.lasterror.get`
Parameters	None
Result	Decimal string see *ODS Get Last Error codes*

6.14. waslv2.lasterror.clear

Description	Clears the last loader error flag to zero.
Command	`waslv2.lasterror.clear`
Parameters	None
Result	“0”

6.15. waslv2.stalledaxis.get

Description	If the SL_LOADER_AXISSTALLED bit is set in the WASLV2 Status Word then this returns the offending axis id.
Command	`waslv2.stalledaxis.get`
Parameters	None
Result	“0”

6.16. waslv2.hotelfitted.get

Description	Determine what hotels are fitted
Command	`waslv2.hotelfitted.get <hotel>`
Parameters	`<hotel>` | Hotel id [1..waslv2.maxhotels.get] | *int*
Result	“0” not fitted, or “1” fitted

6.17. waslv2.trayfitted.get

Description	Determine what trays are fitted
Command	`waslv2.trayfitted.get <hotel> <apartment>`
Parameters	`<hotel>` | Hotel id [1..*ods.maxhotels.get*] | *int* `<apartment>` | Apartment id [1..*ods.maxplatesperhotel.get*] | *int*
Result	“0” not fitted, or “1” fitted

6.18. waslv2.maxhotels.get

Description	Determine the maximum number of supported hotels
Command	`waslv2.maxhotels.get`
Parameters	None
Result	Decimal string representing max hotels count

6.19. waslv2.maxtraysperhotel.get

Description	Determine the maximum number of apartments (trays) in hotel
Command	`waslv2.maxtraysperhotel.get`
Parameters	None
Result	Decimal string representing max hotel apartments (trays)

6.20. waslv2.axis.jog

Description	Used during initialisation and initial setup/calibration to manually jog the loader axis relative to its current position.		
Command	`waslv2.axis.jog <axis> <distance>`		
Parameters	`<axis>`	See *WASLV2 Loader Axes*	*int*
	`<distance>`	Encoder counts	*int*
Result	"0"		

6.21. waslv2.axis.goto

Description	Used during setup/calibration to manually move the loader to a known absolute position. Do not use during initialisation as until initialisation has completed absolute positions are not valid.		
Command	`waslv2.axis.goto <axis> <absolute position>`		
Parameters	`<axis>`	See *WASLV2 Loader Axes*	*int*
	`<absolute position>`	Encoder counts	*int*
Result	"0"		

6.22. waslv2.axis.move-at-velocity

Description	Used during setup/calibration to manually move the loader at a given velocity.		
Command	`waslv2.axis.move-at-velocity <axis> <velocity>`		
Parameters	`<axis>`	See *WASLV2 Loader Axes*	*int*
	`<velocity>`	Encoder counts/s	*int*
Result	"0"		

6.23. waslv2.axis.busy.get

Description	Used to determine whether axis is currently moving. Only needed during setup/calibration when manually moving the loader.
Command	`waslv2.axis.busy.get <axis>`
Parameters	`<axis>` | See *WASLV2 Loader Axes* | *int*
Result	"0" axis idle, "1" axis busy

6.24. waslv2.axis.position.get

Description	Used to determine whether axis is currently moving. Only needed during setup/calibration when manually moving the loader.
Command	`waslv2.axis.position.get <axis>`
Parameters	`<axis>` | See *WASLV2 Loader Axes* | *int*
Result	"0" axis idle, "1" axis busy

6.25. waslv2.calibration.set

Description	Stores the current positions of the loader as the calibrated load/unload position. Only needed during initial stage calibration.
Command	`waslv2.calibration.set`
Parameters	None
Result	"0"

6.26. waslv2.calibration.save

Description	Saves the calibrated positions of the loader into the controller backup and creates a INI file in ProgramData/Prior folder. Only needed during initial stage calibration.
Command	`waslv2.calibration.save`
Parameters	None
Result	"0"

6.27. waslv2.calibration.stagexy.get

Description	Returns the calibrated stage XY position. The application must position the stage to this position before loading or unloading trays to the stage.
Command	`waslv2.calibration.stagexy.get`
Parameters	None
Result	Calibrated stage position in microns from the stage back right limit switch ie "45087,23345"

6.28. waslv2.reloadsetup

Description	Reload the setup with immediate effect on loader positions. Useful during initial calibration when manually tweaking calibrated hotel and stage positions.
Command	`waslv2.reloadesetup`
Parameters	None
Result	“0”

6.29. waslv2.singlestepmode.set

Description	Activate the single step mode of the loader. This is a useful debug facility for stepping through the loaders actions
Command	`waslv2.singlestepmode.set <mode>`
Parameters	`<mode>` | “0” off, “1” on | *int*
Result	“0”

6.30. waslv2.singlestep

Description	With single step mode activated, this command causes the loader to move one-step through its current action state machine. This is a useful debug facility for stepping through the loaders actions such as transferring trays etc
Command	`waslv2.singlestep`
Parameters	None
Result	“0”

6.31. waslv2.serialnumber.get

Description	Return the serial number of the loader
Command	`waslv2.serialnumber.get`
Parameters	None
Result	Serial number of the loader

6.32. waslv2.serialnumber.set

Description	Set the serial number during the setup/calibration process. This value is stored in the INI calibration file.
Command	`waslv2.serialnumber.set <serial>`
Parameters	`<serial>` | Serial number | *int*
Result	“0”

7. APPENDIX

7.1. API Error Codes

SDK name	API Code	Meaning
PRIOR_OK	0	The DLL function call succeeded ok.
PRIOR_UNRECOGNISED_COMMAND	-10001	The requested command was not recognised. Check spelling.
PRIOR_FAILEDTOOPENPORT	-10002	The requested communications port could not be opened. Check port identification and make sure its not already opened by another application.
PRIOR_FAILEDTOFINDCONTROLLER	-10003	The port was opened but no Prior controller found
PRIOR_NOTCONNECTED	-10004	The session is not currently connected to a controller.
PRIOR_ALREADYCONNECTED	-10005	The session is already connected to a controller.
PRIOR_INVALID_PARAMETERS	-10007	Command parameters are incorrect, either incorrect values or number of parameters.
PRIOR_UNRECOGNISED_DEVICE	-10008	The specified ancilliary controller device is not valid. Probably not connected to controller.
PRIOR_APPDATAPATHERROR	-10009	Failure to open file in the application data folder.
PRIOR_LOADERERROR	-10010	A error occurred on the loader in question.Check <loadertype>.lasterror.get
PRIOR_CONTROLLERERROR	-10011	A error occurred on the controller in question. Check controller.lasterr.get
PRIOR_NOTIMPLEMENTEDYET	-10012	Command is valid but not yet implemented.
PRIOR_UNEXPECTED_ERROR	-10100	Something odd happened. Provide Prior with details.
PRIOR_SDK_NOT_INITIALISED	-10200	Call the DLL initalisation routine first before anything else.
PRIOR_SDK_INVALID_SESSION	-10300	An invalid session ID has been specified.
PRIOR_SDK_NOMORE_SESSIONS	-10301	Exceeded session limit of DLL.

7.2. Controller Error Codes
Refer to file PriorScientificSDK.h for full details

7.3. ODS Loader Axes
Refer to file PriorScientificSDK.h for full details

7.4. ODS Loader States
Refer to file PriorScientificSDK.h for full details

7.5. ODS Status Word
Refer to file PriorScientificSDK.h for full details

7.6. ODS Get Last Error codes

Refer to file PriorScientificSDK.h for full details

7.7. SL160 Loader Axes
Refer to file PriorScientificSDK.h for full details

7.8. SL160 Loader States
Refer to file PriorScientificSDK.h for full details

7.9. SL160 Status Word
Refer to file PriorScientificSDK.h for full details

7.10. SL160 Get Last Error codes

Refer to file PriorScientificSDK.h for full details

7.11. WASLV2 Loader Axes
Refer to file PriorScientificSDK.h for full details

7.12. WASLV2 Loader States
Refer to file PriorScientificSDK.h for full details

7.13. WASLV2 Status Word
Refer to file PriorScientificSDK.h for full details

7.14. WASLV2 Get Last Error codes

Refer to file PriorScientificSDK.h for full details