Application Note

Prober High-level Command Set

Wentworth Laboratories

Prober High-level Command Set

This application note is intended as an orientation document for engineers who wish to remotely control Wentworth Labs prober over RS232, GPIB, or Ethernet from a tester or a production control system. It describes the high-level command set, which is common to all our semi-automatic and fully automatic probers. Following this document will ensure that your code will be able to work with all our current and future probers.

ⓘ *This document is intended to supplement the manuals that are shipped with each prober, not to replace them.*

Contents

Introduction 4
- Overview 4
- Command Syntax 4
- Error Reporting 4
- Supported Software Versions 5

Core Commands 6
- $GetProberInfo 6
- $ListRecipes 6
- $SetRecipe / $SetDevice 6
- $ManualWafer 6
- $FirstWafer 7
- $NextWafer 7
- $FirstDie 7
- $GetDieComment 8
- $SetDie 8
- $NextDie 8
- $EndWafer 9
- $AbortRun 9

Hardware Specific Commands 10
- $Lights 10
- $Lamp 10
- $GetTemperature 10
- $SetTemperature 10
- $GetWaferID 11

© 2025 Wentworth Laboratories Ltd. Do Not Reproduce.

$TestDie 11
$InkDie 11
$CleanProbes 11
$GetPos 11
$SetPos 12
$TestHead 12
Specialised Application Commands 13
$MoveDie 13
$SetDieM 13
$ChuckUp 13
$GetWaferFilename 14
$GetVariable 14
$SetVariable 14
$GetInput 14
$SetOutput 15
Appendix A: Example Command Sequences 16
To Test a Single Wafer 16
To Test a Batch of Wafers 16
Appendix B: Example Python Script 17

© 2025 Wentworth Laboratories Ltd. Do Not Reproduce.

Introduction

Overview

This document describes our high-level command set (also known as dollar commands because every command begins with a dollar character), which is common to all our probers.

ⓘ *The command set is called high-level to distinguish it from the AWP (or low-level) command set used by our AWP range of probers that were produced back in the 1990s.*

In the document the commands are split into the following sections:

- Core Commands - This section describes the commands that are available on all probers
- Hardware Specific Commands - This section describes the commands that will be used on probers with specific hardware
- Specialised Application Commands - These commands implement functionality required by specialised applications

In addition, certain systems may have custom commands added to the command set. In this case, the system documentation will describe any such commands.

Command Syntax

For the commands, the following syntax is used:

- All commands are case-sensitive
- **Bold text** indicates a command or a literal parameter
- *Italic text* indicates a parameter value
- Square brackets [ ] indicate an optional parameter (with the default value after the equals sign)
- A bar | indicates a choice between two parameters
- Ellipsis … indicates a repeated parameter
- For clarity in this manual, a middle dot character · denotes a space. Wherever this character appears here, a space must be used in the actual command

Error Reporting

Errors in commands will be reported in the reply to the command, which will be in the following format:

error_id · error_code [: extra_information]

Where:

- error_id is the three-letter error type (normally either PME or INF).
- error_code is the error number.

© 2025 Wentworth Laboratories Ltd. Do Not Reproduce.

- extra_information is optional extra information about the error.

ⓘ *An error code of zero always indicates that the operation was successful (ie, there was no error).*

Supported Software Versions

This high-level command set is fully supported by the following:

- Standalone Pegasus™ probers: Pegasus™ firmware version 7.9 and later.
- Engineering systems: LabMaster FA software version 15.2 and later.
- Production systems: LabMaster Pro software version 12.5 and later.

⚠ *Earlier software versions will have limited or no support. Please contact customer services at Wentworth Laboratories for software upgrade options.*

© 2025 Wentworth Laboratories Ltd. Do Not Reproduce.

Core Commands

This section describes the commands that are available on all probers.

$GetProberInfo

This command returns an information string identifying the prober.

Command Syntax: **$GetProberInfo**

Reply Syntax: *information*

$ListRecipes

This command returns a list of all the available recipes. The recipe names in the list are separated by the vertical bar character.

Command Syntax: **$ListRecipes**

Reply Syntax: *name* | *…*

Usage: This command cannot be used while a wafer is being tested.

Hardware: All probers.

$SetRecipe / $SetDevice

This command changes the active device recipe, which has been trained in local mode. If the recipe_name parameter is omitted, the prober will prompt the user for the recipe.

The alternative syntax is used by Pegasus™ probers without the device or the wafer map facility.

Command Syntax: **$SetRecipe** [· *recipe_name*]

Arguments:

- *recipe_name*: The name of recipe to make the active recipe.

Alternative Command Syntax: **$SetDevice** · *wafer_size* · *step_x* · *step_y*

Arguments:

- *wafer_size*: The wafer size in mm.
- *step_x*: The x step size in mm.
- *step_y*: The y state size in mm.

Usage: This command cannot be used while a wafer is being tested.

Hardware: All probers.

$ManualWafer

This command starts testing a single wafer. It loads and aligns a wafer, but wafer source and the exact alignment procedure will depend on the prober.

© 2025 Wentworth Laboratories Ltd. Do Not Reproduce.

Command Syntax: **$ManualWafer** [· *wafer_source*]

Arguments:

- *wafer_source*: The prober dependent source of the wafer.

Hardware: All probers.

$FirstWafer

This command starts testing the first wafer of a batch of wafers. It loads and aligns a wafer, but wafer source and the exact alignment procedure will depend on the prober.

Command Syntax: **$FirstWafer** [· *wafer_source*]

Arguments:

- *wafer_source*: The prober dependent source of the wafer.

Hardware: All probers.

$NextWafer

This command starts testing the next wafer of a batch of wafers. It loads and aligns a wafer, but wafer source and the exact alignment procedure will depend on the prober.

Command Syntax: **$NextWafer** [· wafer_source]

Arguments:

- *wafer_source*: The prober dependent source of the wafer.

Usage: This command can only be used after a successful $FirstWafer command.

Hardware: All probers.

$FirstDie

This command moves the chuck to the first die position and then lifts the chuck to bring the probe needles into contact with the wafer.

Command Syntax: **$FirstDie** [· *lift_chuck* = 1]

Arguments:

- *lift_chuck*: Whether or not to lift the chuck into contact at the end of the x,y move (with 1 lifting the chuck and 0 not lifting the chuck).

Reply Syntax: 0 , 0

Usage: This command can only be used after a successful $ManualWafer, $FirstWafer, or $NextWafer command.

© 2025 Wentworth Laboratories Ltd. Do Not Reproduce.

Hardware: All probers.

$GetDieComment

This command gets the comment field for the current die from the wafer map. If there is no comment for the current die, then the reply will be a zero-length string. The comment field can be used to store information for the tester.

⚠ *This command is only available in version 2.1 of the API.*

Command Syntax: **$GetDieComment**

Reply Syntax: *comment*

Hardware: Not available on Pegasus™ probers without the wafer map facility.

$SetDie

This command sets the test result for the current die. The test result is a bin number that is stored in the results wafer map, and is normally in the range 1-128. Using this command will also update the wafer map displayed by the prober.

Command Syntax: **$SetDie** · *bin_number*

Arguments:

- *bin_number*: The new bin number.

Hardware: Not available on Pegasus™ probers without the wafer map facility.

$NextDie

This command moves the chuck to the position of the next die to be tested and then lifts the chuck to bring the probe needles into contact with the wafer.

An error is returned to indicate that the end of the wafer has been reached. The exact error code will be prober specific (see the prober manual for more information).

Command Syntax: **$NextDie** [· *lift_chuck* = 1]

Arguments:

- *lift_chuck*: Whether or not to lift the chuck into contact at the end of the x,y move (with 1 lifting the chuck and 0 not lifting the chuck).

Reply Syntax: *die_x* , *die_y*

Usage: This command can only be used after a successful $FirstDie or $NextDie command.

© 2025 Wentworth Laboratories Ltd. Do Not Reproduce.

Hardware: All probers.

$EndWafer

This command ends testing a wafer. It unloads the wafer to the manual load position. The exact unload procedure will depend on the prober.

Command Syntax: **$EndWafer**

Usage: This command should be used after a $NextDie command returns an error to indicate that the end of the wafer has been reached.

Hardware: All probers.

$AbortRun

This command aborts testing a wafer. It unloads the wafer to the manual load position. The exact unload procedure will depend on the prober.

Command Syntax: **$AbortRun**

Hardware: All probers.

© 2025 Wentworth Laboratories Ltd. Do Not Reproduce.

Hardware Specific Commands

This section describes the commands that will be used by probers with specific hardware.

$Lights

This command turns on/off the red, amber, and green lights on a traffic light pole. This command might to be used to indicate to the operator that a wafer has finished testing.

Command Syntax: **$Lights** · *rag*

Arguments:

- *rag*: the new red/amber/green lights (with 1 for on and 0 for off).

Hardware: Probers with an integrated traffic light pole.

$Lamp

This command turns on/off the microscope lamp. This command might be used to turn the microscope lamp off before light sensitive devices are tested.

Command Syntax: **$Lamp** · *state*

Arguments:

- state: the new state (either ON or OFF).

Hardware: Probers with an integrated microscope lamp.

$GetTemperature

This command returns the current temperature (in degrees C) of the hot chuck.

Command Syntax: **$GetTemperature**

Reply Syntax: *temperature*

Hardware: Probers with an integrated hot chuck.

$SetTemperature

This command starts changing the temperature value (in degrees C) of the hot chuck.

ⓘ *This command returns immediately and the $GetTemperature command should be used to detect when the temperature change is complete.*

Command Syntax: **$SetTemperature** · *temperature*

Arguments:

- *temperature*: The new temperature value in degrees C.

© 2025 Wentworth Laboratories Ltd. Do Not Reproduce.

Hardware: Probers with an integrated hot chuck.

$GetWaferID

This command returns the wafer ID of the wafer being tested.

Command Syntax: **$GetWaferID**

Reply Syntax: *ID*

Usage: This command can only be used after a successful $ManualWafer, $FirstWafer, or $NextWafer command.

Hardware: Probers with an integrated wafer ID reader (an OCR reader, 2D data matrix or barcode reader).

$TestDie

This command tells the prober to test the current die using the currently selected tester.

Command Syntax: **$TestDie**

Reply Syntax: *bin_number*

Hardware: Probers with a simple TTL tester connected directly to the controller unit.

$InkDie

This command tells the prober to ink the current die. This command is ignored if the probers doesn't have an inker.

Command Syntax: **$InkDie**

Hardware: Probers with an integrated inker.

$CleanProbes

This command tells the prober to clean the probes.

Command Syntax: **$CleanProbes**

Hardware: Probers with an integrated test probe cleaner hardware.

$GetPos

This command returns the current x,y,z position of a device.

Command Syntax: **$GetPos** · *device*

Arguments:

- *device*: Can be one of Chuck, PMM, CAP1, CAP2, CAP3, CAP4, CAP5, CAP6.

Reply Syntax: *x_position* , *y_position* , *z_position*

© 2025 Wentworth Laboratories Ltd. Do Not Reproduce.

Hardware: The given device must be fitted to the prober.

$SetPos

This command sets x,y,z position of a device. If the device is not present, then an error is return.

Command Syntax: **$SetPos** · *device* · *x_position* · *y_position* [· *z_position*]

Arguments:

- *device*: Can be one of Chuck, PMM, CAP1, CAP2, CAP3, CAP4, CAP5, CAP6.
- *x_position*: The new x-axis position in mm.
- *y_position*: The new y-axis position in mm.
- *z_position*: The new z-axis position in mm.

Hardware: The given device must be fitted to the prober.

$TestHead

This command tells the prober to select the given test head.

Command Syntax: **$TestHead** · *head_number*

Arguments:

- *head_number*: the number of the test head.

Hardware: Probers with an integrated test head hardware.

© 2025 Wentworth Laboratories Ltd. Do Not Reproduce.

Specialised Application Commands

These commands implement functionality required by specialised applications.

$MoveDie

This command brings the probe needles into contact with the specified die.

Command Syntax: **$MoveDie** · *die_x* · *die_y* [· *lift_chuck* = 1]

Arguments:

- *die_x*: The x die position (relative to the home die).
- *die_x*: The y die position (relative to the home die).
- *lift_chuck*: Whether or not to lift the chuck into contact at the end of the x,y move (with 1 lifting the chuck and 0 not lifting the chuck).

Usage: This command can only be used after a successful $ManualWafer command.

Hardware: All probers.

$SetDieM

This command replaces the $SetDie command when multiple die are being tested simultaneously.

Command Syntax: **$SetDieM** · *bin_number* · …

Arguments:

- *bin_number*: The new bin number.

Hardware: Not available on Pegasus™ probers without the wafer map facility.

$ChuckUp

This command raises or lowers the chuck.

ⓘ *This command is not normally required because the $FirstDie, $NextDie, and $MoveDie commands bring the wafer into contact with the probe needles.*

Command Syntax: **$ChuckUp** · *lift_chuck*

Arguments:

- *lift_chuck*: 1 to raise the wafer into contact with the probe needles, or 0 to lower the wafer from contact.

Hardware: All probers.

© 2025 Wentworth Laboratories Ltd. Do Not Reproduce.

$GetWaferFilename

This command is used to get the path of the results wafer map file. The path file will vary between prober types (see the prober user manual for details).

Command Syntax: **$GetWaferFilename**

Reply Syntax: *filename*

Hardware: Not available on Pegasus™ probers without the wafer map facility.

$GetVariable

This command returns the value of a prober specific variable (see the prober user manual for a list of the available variables).

ⓘ *All probers support the RecipeName variable, which is the name of the active device recipe.*

Command Syntax: **$GetVariable** · *name*

Arguments:

- *name*: The name of the variable.

Reply Syntax: *value*

Hardware: All probers.

$SetVariable

This command changes the value of a prober specific variable (see the prober user manual for a list of the available variables).

ⓘ *Not all variables can be changed at any time: Some variables may not be able to be changed while testing a wafer, and other variables may not be able to be changed at any time (ie, they are read-only variables).*

Command Syntax: **$SetVariable** · *name* · *value*

Arguments:

- *name*: The name of the variable.
- *value*: The new value of the variable.

Hardware: All probers.

$GetInput

This command returns the state of an input line on the prober (see the prober user manual for a list of the available input lines).

Command Syntax: **$GetInput** · *line_number*

© 2025 Wentworth Laboratories Ltd. Do Not Reproduce.

Arguments:

- *line_number*: the input line number.

Reply Syntax: *state*

Hardware: All probers.

$SetOutput

This command changes the state of an output line on the prober (see the prober user manual for a list of the available output lines).

⚠ *Changing the state of an output line could cause hardware damage.*

Command Syntax: **$SetOutput** · *line_number* · *state*

Arguments:

- *line_number*: the output line number.
- *state*: the new state (1 for high and 0 for low).

Hardware: All probers.

© 2025 Wentworth Laboratories Ltd. Do Not Reproduce.

Appendix A: Example Command Sequences

This appendix has a couple of example command sequences. In these examples the commands are indicated in bold and the actions to be performed by the control system are indicated in italic.

⚠ *For clarity only minimal error handling is included in these examples. In production code all command replies must be checked for errors and the appropriate action taken if an error occurs.*

To Test a Single Wafer

This shows the minimum command sequence to test a single wafer.

$SetRecipe wafer_type

$ManualWafer

$FirstDie

While no error returned

    Perform test

    $NextDie

End while

$EndWafer

To Test a Batch of Wafers

This shows the minimum command sequence to test a batch of wafer.

$SetRecipe wafer_type

$FirstWafer

While no error returned

    $FirstDie

    While no error returned

        Perform test

        $NextDie

    End while

    $NextWafer

End while

$EndWafer

© 2025 Wentworth Laboratories Ltd. Do Not Reproduce.

Appendix B: Example Python Script

Below is an example command script to test a single wafer using high-level command being set over RS232. It is written in Python 3 and assumes the pyserial module has been installed. The code is minimal and does not have a user interface or any error handling and is therefore *not suitable for Production use*. This sample code is provided "as is" with no guarantee.

```
import serial, sys, time

#-------------------------------------------------------------------------------
def TestWafer(serial_port):
	"""
	Tests a single wafer

	Args:
		serial_port: the serial port object

	Returns: whether the wafer test completed successfully
	"""

	print("*** Please follow the instructions on the prober's screen ***")
	reply = SendCommand(serial_port, "$ManualWafer")
	if not reply[0]:
		print("Wafer aborted by operator")
		return False

	reply = SendCommand(serial_port, "$FirstDie")
	while reply[0]:  # Command successful
		print(f"Testing die at {reply[2]}...")
		# Replace next line with test code
		bin_number = 2
		# Update the prober's wafer map (optional)
		reply = SendCommand(serial_port, f"$SetDie {bin_number}")

		# $NextWafer returns an error at end of wafer
		reply = SendCommand(serial_port, "$NextDie")

	print("*** Please follow the instructions on the prober's screen ***")
	SendCommand(serial_port, "$EndWafer")

	print("Wafer completed")
	return True
#-------------------------------------------------------------------------------
def SendCommand(serial_port, command):
	"""
	Sends a command to the prober and waits for the reply

	Args:
		serial_port: the serial port object
		command: the command text

	Returns: a tuple containing whether the command was successful, the
		prober's status byte, and the reply text.
	"""

	serial_port.write((command + "\n").encode("ascii"))
```


© 2025 Wentworth Laboratories Ltd. Do Not Reproduce.

```
        this_char = ""; reply = ""
        while this_char != "\n":
                while serial_port.inWaiting() == 0:
                        time.sleep(0.1)
                this_char = serial_port.read(1).decode("ascii")
                if this_char != "\n":
                        reply += this_char

        status_byte = int(reply[:2], 16)
        success = status_byte < 0x80
        text = "" if len(reply) <= 2 else reply[2:]
        return (success, status_byte, text)
#------------------------------------------------------------------------------
if __name__ == "__main__":
        port_name = "COM1" if len(sys.argv) == 1 else sys.argv[1]

        serial_port = serial.Serial(
                port_name,
                baudrate=38400,
                parity=serial.PARITY_EVEN,
                bytesize=serial.SEVENBITS,
                stopbits=serial.STOPBITS_TWO,
                timeout=300,
                writeTimeout=10
        )
        if serial_port.isOpen():
                TestWafer(serial_port)
                serial_port.close()
#------------------------------------------------------------------------------
```


© 2025 Wentworth Laboratories Ltd. Do Not Reproduce.