



**THIS SERVICE MANUAL IS ISSUED IN ADOBE ACROBAT FORMAT FOR EASE OF VIEWING AND DISTRIBUTING AND CAN BE VIEWED AND PRINTED AS REQUIRED.**

**A LIST OF CHAPTERS IS SHOWN ON THE LEFT HAND SIDE OF THE SCREEN. CLICKING ON AN ARROW WILL DISPLAY THE CONTENTS OF THE CHAPTER. TO VIEW THE REQUIRED CHAPTER OR SECTION, CLICK ON THE TEXT OF THE REQUIRED HEADING.**

**THE LIST CAN BE HIDDEN BY CLICKING THE LEFT ICON ON THE TOOLBAR AND WILL BE DISPLAYED BY CLICKING ON THE SECOND TO LEFT ICON.**

**TO MOVE THROUGH THE MANUAL, USE THE SCROLLBAR OR THE SINGLE ARROWS IN THE TOOLBAR (PAGE UP AND PAGE DOWN). THE DOUBLE ARROWS ON THE TOOLBAR MOVE TO THE FIRST AND LAST PAGES.**

**THE PAGE CAN BE ZOOMED IN BY CLICKING ON THE MAGNIFYING GLASS IN THE TOOLBAR. THIS TOOL WILL ALSO ZOOM OUT (e.g. WHEN VIEWING CIRCUIT DIAGRAMS) IF THE CTRL KEY IS PRESSED WHEN CLICKING ON THE PAGE.**

**THE PAGE WILL BE EXPANDED IF THE RIGHT PAGE LAYOUT ICON IS CLICKED JUST TO THE LEFT OF THE SEARCH BINOCULARS ICON. THE FULL PAGE WILL BE DISPLAYED IF THE MIDDLE PAGE ICON IS CLICKED.**

**SOME OF THE TEXT IN THE MANUAL IS COLOURED:- CLICKING ON BLUE TEXT WILL MOVE TO A LOCATION IN THE SAME CHAPTER AND CLICKING ON RED TEXT WILL MOVE TO A LOCATION IN A DIFFERENT CHAPTER.**

**TO RETURN TO THE ORIGINAL POSITION AFTER MOVING THROUGH A LINK, CLICK THE RIGHT MOUSE BUTTON THEN SELECT GO BACK.**

**THIS PROCEDURE WILL ALWAYS MOVE BACK THROUGH THE SCREENS EVEN THOUGH A LINK WAS NOT USED.**

---

# **NEXYGEN Ondio**

**USER MANUAL**

*Copyright Lloyd Instruments Ltd. 2000*

# LLOYD INSTRUMENTS LTD

Forum House  
12 Barnes Wallis Road  
Segensworth East  
Fareham  
Hampshire  
PO15 5TT

Tel: +44 (0) 1489 574221

Fax: +44 (0) 1489 885118

E-mail:

[techsupport@lloyd-instruments.co.uk](mailto:techsupport@lloyd-instruments.co.uk)

March 2000.

User Manual	Part No: 01/2865
	Version: 4.0
For Software	Part No: 40/0683
	Version: 4.0
	Part No: 40/0695
	Version: 4.0



W.D. Turner Company, Inc.

*Sales & Technical Services*

#7 Whitaker Place  
Thomasville, North Carolina 27360

**Phone: (336) 882-4931**

Fax: (336) 882-5175

[www.WDturner.com](http://www.WDturner.com)

## NOTICE

This Manual or Spec Sheet was provided through our web site to assist Customers who purchased equipment through W. D. Turner Company, Inc.

**Please do not operate, repair or modify equipment without proper training.** Our company provides on-site calibrations, repairs and training. Please contact our office for free quotes or more information.

Please use this Manual or Spec Sheet at your own risk. We welcome inquires about this equipment. Telephone and email technical support is provided free for our customers.

You are invited to visit our web site at:

[www.wdturner.com](http://www.wdturner.com)

Thank you for considering our products and services!

# CONTENTS

<b>INTRODUCTION</b>	<b>11</b>
1.1 LICENCE AGREEMENT	11
1.2 TickIT CERTIFICATION OF <b>NEXYGEN</b>	12
1.3 INTRODUCTION TO ONDIO	14
<b>REQUIREMENTS</b>	<b>18</b>
2.1 RECOMMENDED COMPUTER SYSTEM	15
2.2 MINIMUM REQUIRED COMPUTER SYSTEM	15
2.3 OPERATING SYSTEMS	15
2.4 OTHER SOFTWARE	15
<b>INSTALLATION</b>	<b>16</b>
3.1 INSTALLATION FROM CDROM	16
3.2 STARTING THE SETUP	16
3.3 MANUAL INSTALLATION	16
<b>TUTORIAL</b>	<b>18</b>
4.1 STARTING AN ONDIO TEST	18
4.2 NEW TEST ASSISTANT	19
4.3 THREE PAGES	20
4.4 THE FIRST PAGE: TEST PARAMETERS	20
4.5 DIRECTION	20
4.6 PRELOAD	21
4.7 HEIGHT	21
4.8 AREA	22
4.9 BREAK	22
4.10 AUTO-ZERO	23
4.11 AUTO-RETURN	23
4.12 EXTRA RESULTS	23
4.13 THE SECOND PAGE: PRIMARY SCRIPT	23
4.14 THE STAGE COMMAND	24
4.15 PARAMETERS FOR THE STAGE COMMAND	25
4.16 SQUARE BRACKETS	25
4.17 THE RESULT COMMAND	26
4.18 THE LOADATMAX PROPERTY	26
4.19 THE WORK PROPERTY	27
4.20 RUNNING THE TEST	27
4.21 PARAMETERS WITH UNITS	27
4.22 ADDING A RESULT	28
4.23 CALCULATIONS	29
4.24 LOAD LIMITS	30
4.25 LOAD RATES	30

4.26	A CYCLE TEST	31
4.27	MANY CYCLES	31
4.28	OTHER CYCLE LIMITS	32
4.29	USING THE LOOP COUNTER	32
4.30	CALCULATING STAGE LIMITS	33
4.31	TEST PROGRESS	34
4.32	ADVANCING A TEST	34
4.33	ADVANCING A LONG TEST	35
4.34	THE DIFFERENCE BETWEEN END AND ADVANCE	36
4.35	A MODULUS TEST	38
4.36	A MORE ACCURATE MODULUS TEST	39
4.37	THE STATIC LOAD FUNCTION	41
4.38	A LIMITATION OF CYCLE TESTS	41
4.39	ANOTHER WAY TO BEAT PAUSES	42
4.40	THE RUNALLSTAGES FUNCTION	45
4.41	THE SECONDARY SCRIPT	45
4.42	MARKERS	46
4.43	TEST-GLOBAL VARIABLES	47
4.44	AN AUTOMATIC MODULUS TEST	48
4.45	AUTOMATIC OFFSET YIELD	48
4.46	BREAK RESULTS	49
4.47	USING THE BREAK DETECTOR IN A CYCLE TEST	50

## **DATA ACQUISITION 55**

5.1	INTRODUCTION	51
5.2	DEVICE SETTINGS	51
5.3	ANALOGUE INPUTS	53
5.4	EXAMPLES	54

## **REFERENCE 55**

6.1	PRIMARY SCRIPT PROPERTIES	55
6.2	PRIMARY SCRIPT FUNCTIONS	56
6.3	SECONDARY SCRIPT PROPERTIES	58
6.4	SECONDARY SCRIPT FUNCTIONS	58
6.5	A TABLE OF UNITS	60
6.6	ALPHABETICAL REFERENCE	64

---

## **ADVANCED SCRIPTS 93**

7.1	THE VBSCRIPT LANGUAGE	93
7.2	UNIT-TRACKING EXTENSIONS	94
7.3	OTHER SCRIPT LANGUAGES	95
7.4	STOPPING THE MACHINE	97

---

<b>END USER LICENCE AGREEMENT</b>	<b>98</b>	
8.1	DEFINITIONS	98
8.2	LICENCE	98
8.3	REPRODUCTION RIGHTS	98
8.4	SECURITY AND TRANSFERABILITY	99
8.5	TITLE	99
8.6	WARRANTY	99
8.7	LIABILITY	100
8.8	TERMS AND TERMINATION	100
8.9	GENERAL	101

---

---

## 1.1 LICENCE AGREEMENT

Enclosed is **NEXYGEN** Ondio. Please check that you have the following items and read the Licence Agreement on page 98 BEFORE opening the sealed security device. You should have the following:

- One **NEXYGEN** Ondio CDROM, Part No. 40/0694.
- One **NEXYGEN** Ondio Manual, including Licence Agreement
- One Licence Registration Form

Ondio is an add-on component to enhance the standard **NEXYGEN**.

Under the terms of the Licence Agreement you are allowed to make a backup copy of this material only for the purpose of protecting its integrity in the event of the working copy becoming damaged or corrupted.

You will note that your disk and Licence Registration Form is issued with a Licence Number. This is used by Lloyd Instruments Limited as a cross reference to you, the customer, and to the particular version of software that you have purchased. This number must be quoted in any correspondence relating to this software, therefore you must complete the Licence Registration Form and return it to the Software Manager, Lloyd Instruments Limited, in order that we can give you the technical support you may initially require.






# 1.0 INTRODUCTION

1.2

## TICKIT CERTIFICATION OF **NEXYGEN**

The **NEXYGEN** range of software has been designed in accordance with the requirements of our TickIT registration.

 	Lloyd instruments Ltd. 12 Barnes Wallis Road Segensworth East Fareham. UK PO15 5TT	 A trademark of AMETEK, Inc.
<b><u>NEXYGEN Software</u></b>		
<p>The <b>NEXYGEN</b> range of software products has been designed in accordance with the requirements of our TickIT * registration. (Certificate no. Q11777)</p>		
<p>These products have been developed by trained employees within Lloyd Instruments Ltd.</p>		
<p>The software has been produced to Lloyd Instruments' own design specification and is intended to be fully compatible with a defined range of Lloyd machine products.</p>		
<p>Design planning has ensured that responsibility for each phase of design and development was clear, and the appropriate personnel and resources were assigned.</p>		
<p>Regular design reviews were conducted to ensure that all aspects of Customer needs, performance, reliability, cost and safety etc. were considered.</p>		
<p>Design verification and validation was performed incorporating tests, demonstrations, calculations and code reviews to confirm that the requirements of the design specification were achieved.</p>		
Page 1		



# 1.0 INTRODUCTION

Full traceability of design changes was maintained throughout the development cycle. Security of master versions being ensured through full back-ups and archiving.

The software has been independently tested, prior to release, by Applications Engineers to ensure that the performance of the software meets the acceptance criteria, performs accurate calculations, is not affected by combinations of unusual events, and in the case of unforeseen failure, will fail-safe. These Engineers are available to give Customer support and training, as required.

External auditing of the software development process is performed by software specialists from SGS Yarsley International Certification Services Ltd. Internal auditing is conducted under the direction of the Quality Manager, as documented in the Quality system.

Signed : ..... Date : .....  
M G Bryant – Quality Manager

- \* *TickIT registration relates directly to ISO 9001 : 1994 and is based on the additional guidance material in :*
- \* *ISO 9000-3 'Guidelines for the application of ISO 9001 to the development, supply and maintenance of software'.*
- \* *ISO/IEC 12207 'Information technology – Software life-cycle processes'.*
- \* *The TickIT guide Issue 4.0*

Congratulations on your purchase of the Ondio software. This introduction provides an overview of the philosophy and operation of Ondio.

Ondio is an add-on for **NEXYGEN** that enhances its capabilities with:

- Tests which involving multiple stages.
- Cycle tests.
- Tests which involve load rate and load holding stages, where supported by your materials testing machine.
- User-defined results calculations.
- External data capture via a National Instruments Data Acquisition Card.

It is recommended that you read through the **NEXYGEN** manual before this manual for Ondio. The **NEXYGEN** manual contains important information about configuring your machine, and using general features of **NEXYGEN** used for organising your data storage and analysing results. All of these features are equally important for effective use of Ondio.

This manual contains a tutorial which describes the features of Ondio. Each feature is introduced and described in several easy steps.

Having read both tutorials, you should be able to perform customised, sophisticated tests with ease.

## 2.0 REQUIREMENTS

---

### 2.1 RECOMMENDED COMPUTER SYSTEM

- 400MHz Pentium II Processor
- 32MB RAM
- 1 Free COM Port, with a 16550 UART
- CDROM for installation

---

### 2.2 MINIMUM REQUIRED COMPUTER SYSTEM

- 166 MHz Pentium Processor. A faster processor may be required for more complex tests.
- 16MB RAM. More memory may be required for some types of test.
- 20MB Hard Disk Space
- 1 Free COM Port, with a 16550 UART
- CDROM for installation

---

### 2.3 OPERATING SYSTEMS

- Windows 95 or later

---

### 2.4 OTHER SOFTWARE

- Ondio Version 4.0 requires **NEXYGEN** Version 4.0, also from Lloyd Instruments. This version of Ondio may not work with other versions of **NEXYGEN**. If you upgrade either product, you will need to upgrade the other to the same version number.
- The 'Batch Report' feature of **NEXYGEN** requires Microsoft Word Version 7, or later.

## 3.0 INSTALLATION

---

### 3.1 INSTALLATION FROM CDROM

If you received **NEXYGEN** and Ondio on CDROM, you need only insert the disks into your CDROM drive and the setup programs will automatically begin.

If the setup program does not automatically run, open the CD ROM drive window in My Computer and double click the file setup.exe.

You should insert the **NEXYGEN** disk first, and finish that installation process before inserting the Ondio disk.

---

### 3.2 STARTING THE SETUP

After installing **NEXYGEN** you can proceed to install the Ondio add-on.

The setup program contains step-by-step instructions to guide you through the setup process, the first step being a choice of directory.

**Note:** You must enter the same directory as was used to install **NEXYGEN**.

The installation program will suggest the directory Program Files\**NEXYGEN**. You will not need to change this unless you also changed the directory when installing **NEXYGEN**.

---

### 3.3 MANUAL INSTALLATION

Ondio may need to update some components of your operating system with more recent versions. All updates occur automatically, although at some points the installation procedure may require you to restart your computer, and rerun the installation program so that it can continue.

The installation procedure will inform you of what it is about to do before making any changes, allowing you to

## 3.0 INSTALLATION

confirm decision. You may be presented with the following options

### VBSCRIPT INSTALLATION

Ondio uses the VBScript programming language distributed by Microsoft. If you are using Windows 95 or Windows NT 4 then the VBScript system may need to be updated.

If you are using Windows 98 (or later), or if you have Internet Explorer 4 installed, then it is likely that this will not need to be changed.

If the installation process detects a version earlier than 3.1 then it will install an update. This installation process also installs the JScript language at the same time, but this is not normally used by Ondio.

If you wish to install the update manually, it can be found in the directory Microsoft\VBScript\scr31en.exe on the Ondio CDROM.

### DCOM INSTALLATION

VBScript 3.1 requires an update to the DCOM system if you are using Windows 95.

If you are using Windows 98 (or later) or Windows NT, then it is likely that this will not need to be changed.

If you wish to install the update manually, it can be found in the directory Microsoft\DCOM\dcom95.exe on the Ondio CDROM. You will need to install this (and restart your computer) **before** installing VBScript

### UNINSTALLATION

Once Ondio has been installed, running the setup program again will activate the uninstaller. This differs from previous versions which used a separate uninstall program. Thus to reinstall Ondio to update the files, you should uninstall it first.

## 4.0 TUTORIAL

This tutorial is a step by step guide to the features of **NEXYGEN** Ondio. You will need to spend a few minutes with this guide before you start your own tests.

---

### 4.1

#### STARTING AN ONDIO TEST

Take the following steps to start a new **NEXYGEN** Batch Document containing an Ondio Test Setup.

##### CREATE A NEW BATCH

Create the Batch Document by right-clicking in a folder window, and choose the New **NEXYGEN** Batch Document command.

##### CHOOSE A CATEGORY

You will have to choose a category of test from the list. **NEXYGEN** Ondio is under the "Uncategorised" heading.

##### CHOOSE **NEXYGEN** ONDIO

Choose **NEXYGEN** Ondio from the list of test types.

##### ENTER FILENAME

Enter the name of the file to store this Batch of tests.

##### OPEN THE BATCH DOCUMENT

You should now see your new **NEXYGEN** Batch Document in the folder. Double-click on it to open it. This will show you an empty table of results.

##### OPEN THE FIRST TEST

Open the first test using either the Insert New Test command, or double-clicking on the first row that has already been prepared for you.

## 4.0 TUTORIAL

### OPEN THE ONDIO TEST SETUP

Like all test setup editors in **NEXYGEN**, the test setup information is stored in a small box paperclipped to the graph. Unlike most, the box for Ondio just contains the Ondio logo: There is too much detail in an Ondio test setup to fit all the information in a small box. However, double-clicking on it will cause it to open in a separate window.

---

#### 4.2

### NEW TEST ASSISTANT

When you first open a new Ondio test you will be offered a choice of several different starting points. You may find it easier to set up your first few tests if you choose the starting point closest to your needs.

The choices are:

- A Multi-Stage test. Choose this option if you need to move the machine to different positions, at different speeds.
- A Creep test. Choose this option if you need to hold a force on a sample for a fixed time and measure the change in deflection.
- A Timed Cycle test. Choose this option if you need to move the machine between different positions for a fixed time, or until some other condition is reached.
- A Fixed Cycle test. Choose this option if you need to move the machine between different positions, and repeat the cycle a fixed number of times.
- A blank sheet. You might find this more convenient when you are familiar with Ondio's operation.

Remember that these are just starting points. If you need to do something that is not on the list, or perhaps you need elements from two different things on the list, then just pick the starting point that is closest to what you need.

If you are working through this tutorial with a computer, then choose the Multi-Stage test now.

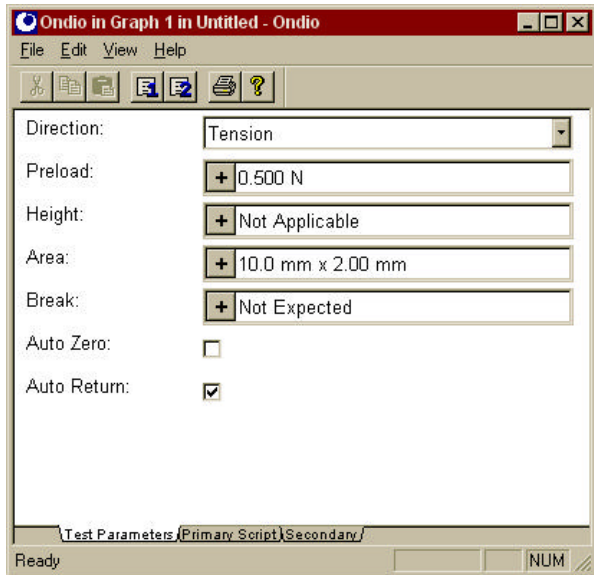
## 4.0 TUTORIAL

### 4.3 THREE PAGES

The Ondio window is made up of three pages. In Figure 1 you can see the first page, and the three tabs along the bottom edge of the window that you can use to switch between the three pages.

Each page contains a different type of information used to set up your test.

### 4.4 THE FIRST PAGE: TEST PARAMETERS



**Figure 1**

The first page, labelled "Test Parameters", lists all of the parameters which are common to all Ondio tests.

### 4.5 DIRECTION

The default setting is "Tension", where all force and position measurements are taken in an upwards direction. Alternatively choose "Compression", where all measurements are in a downwards direction.

## **4.0 TUTORIAL**

Note that this choice does not restrict what types of test you can perform. You can apply both compressive and tensile forces during a test by using negative numbers.

---

### **4.6 PRELOAD**

This drop-down box specifies whether the machine will apply a force on the sample at the start of the test. This is useful for automatically compensating for sample and grip slack.

If you use a preload then you must specify the force which the machine will apply, and the speed at which it should approach the sample. The Extension axis will be zeroed when this force is reached.

If you do not use a preload then the Extension axis zero is the same as the machine zero.

---

### **4.7 HEIGHT**

This drop-down box specifies how the sample height is to be obtained.

#### **AUTO-MEASURE HEIGHT**

This option is only available for compression tests. If it is chosen the machine will automatically measure the height of each sample. At the start of your first test the machine will prompt the operator to clear the machine, and it will drive the crosshead plates together at the specified speed and force. At the start of every test it will prompt the operator to load the sample, then measure its height by driving the crosshead onto the top of the sample using the same speed and force.

#### **KNOWN HEIGHT**

The height of the sample should be entered in the box provided. If this is a nominal value that will be measured precisely before each test, check the box provided.

### 4.8

#### AREA

This drop-down box specifies how the sample area is to be obtained. Choose from the range of cross-sections, and enter the appropriate dimensions.

If these dimensions are nominal values that will be measured precisely before each test, check the box provided.

---

### 4.9

#### BREAK

It is possible to configure Ondio to react when the sample breaks. In most cases this reaction will be to end the test and take a reading, although other more sophisticated possibilities are available, all of which are described later in this tutorial.

Use this option to specify the characteristics which constitute a break.

#### **BREAK NOT EXPECTED**

Use this option if you do not expect the test to break, or if you do not want the Ondio test to react even if it does.

#### **BREAK WHEN THE LOAD DROPS SHARPLY**

Use this option if you are expecting the load to drop sharply at the break point, possibly after decreasing gradually. This option is useful for materials such as metals and plastics which have a quick clean break.

#### **BREAK WHEN THE LOAD DROPS TO A PERCENTAGE OF MAXIMUM LOAD**

Use this option if you are not expecting the load to drop sharply. This option is useful for materials such as timber which have a gradual break, or for materials where the load drops sharply before the real break point.

Enter the percentage of the maximum load which should trigger the break detector. For example if you enter 20%

## 4.0 TUTORIAL

and the load rises to 50N, the break detector will not trigger until the load drops back to 10N.

In some cases you may see some drops in load at the start of the test due to the sample slipping or settling in the grips. In many cases this drop would be a sufficiently high percentage to count as a break (in the example above, the load might rise to 0.5N then drop back to 0.1N). A box is provided for you to enter the load at which the break detector is turned on, in which you should enter the load at which all slippage has finished. As a general rule this should be twice as big as your preload force.

---

### 4.10 AUTO-ZERO

If this option is turned on then the machine load and extension will be zeroed before the start of each test.

---

### 4.11 AUTO-RETURN

If this option is turned on then the machine will return to it's zero extension position after finishing each test.

---

### 4.12 EXTRA RESULTS

Like all **NEXYGEN** test setups, Ondio allows you to specify extra questions which will be asked of the operator before or after the test. Use the Extra Results commands on the Edit menu to set up the questions, and they will be asked automatically.

---

### 4.13 THE SECOND PAGE: PRIMARY SCRIPT

The information that you enter on the first page determines the actions that are taken at the start of the test and at the very end. The second page, the Primary Script, is where you specify what happens *during* the test.

This Primary Script controls where the machine moves, what readings are taken, and what results presented. The commands that you can put in this script give you complete

## 4.0 TUTORIAL

control over all aspects of the test, and the rest of this tutorial covers each command in turn.

You may have noticed that there is a third page: the Secondary Script. This will be explained later, and for these first examples you can leave it blank.

The primary script page may look similar to Figure 2. You can make it look exactly like this example by creating a new Batch Document, and choosing the Multi-Stage test starting point.

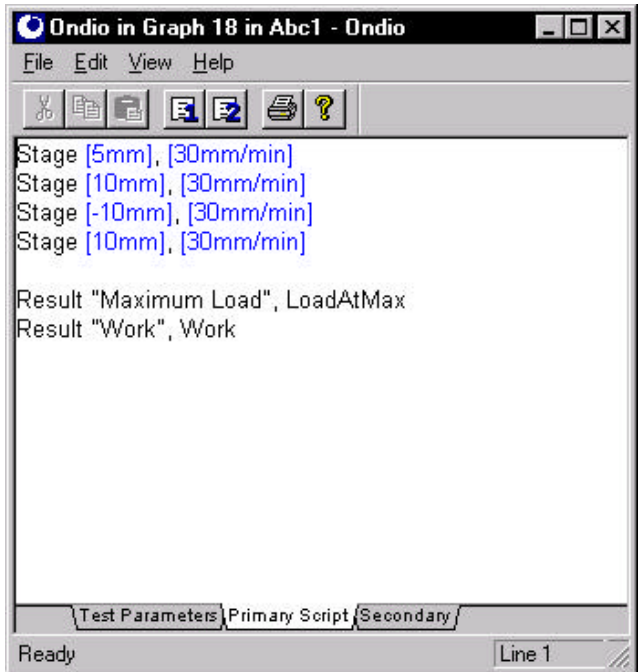


Figure 2

---

### 4.14 THE STAGE COMMAND

The first four lines of this example use the **stage** command to move the machine to a specific position at a specific speed. The machine moves to the four locations in the order specified in the script.

## 4.0 TUTORIAL

Here notice that one of the positions uses a negative number. The machine will move both above and below its zero position.

---

### 4.15 PARAMETERS FOR THE STAGE COMMAND

Following the word **stage** are two parameters, separated by a comma. The first parameter is the limit of the stage, in this example the limit is an extension value measured from either the preload or the machine zero, depending on the parameters set for the preload as explained on page 21.

The second parameter is the speed. The speed should always be above zero.

---

### 4.16 SQUARE BRACKETS

You may notice that both parameters, the limit and the speed, are surrounded by square brackets. The square brackets are needed because all of these parameters (positions and speeds) require values with units. The square brackets indicate to Ondio that it should automatically keep track of units.

If you use square brackets in a script then Ondio will automatically highlight the value in a blue color to indicate that it understands that unit

If you use a unit that it does not understand, or make a spelling mistake in the unit, then it will be highlighted in red.

Here are some examples of valid values with units:

[1.0N]	[0.05lbf]	[3m]
[10microns]	[6in]	[6mm/min]
[0.0001m/s]	[5kN/s]	[5MPa]
[0.1mm/s]	[-5mm]	[20]

Here are some example of invalid values. All of these would be highlighted in red.

## 4.0 TUTORIAL

[mm]	You have the units, but no value.
[6inches]	The spelling of the units is wrong. It should be [6in].
[100n]	The units must be in the correct case. This should be [100N]
[0.04lbf	Missing the closing square bracket
0.04lbf	Missing both square brackets
[ 1.0N]	You can not have spaces anywhere
[1 000 000 N]	between the square brackets
[10\$]	Ondio only understands scientific and engineering units

---

### 4.17 THE RESULT COMMAND

The last two lines of the script use the **Result** command to put a value onto the table of results.

The first parameter is the title of the result. This is used as the column heading. It must be a text value (a string), and therefore must be surrounded by double quote marks.

The second parameter to the **Result** command is the value. In these two cases the values are the built-in properties **LoadAtMax** and **Work**.

Note that you do not need to use square brackets here since both of these built-in properties are already values with units.

---

### 4.18 THE LOADATMAX PROPERTY

The **LoadAtMax** built-in property contains the maximum force over the whole test so far.

Two complimentary properties store other information about the maximum load point: **ExtnAtMax** stores the

## 4.0 TUTORIAL

extension at that point, and **WorkToMax** store the work done on the sample up to that point.

---

### 4.19 THE WORK PROPERTY

The **Work** built-in property contains the Work done on the sample up to the end of the previous stage.

Again, there are two complementary properties **Load** and **Extn** that store the load and extension values at the same point.

---

### 4.20 RUNNING THE TEST



To run this test you need to close the Ondio window, return to the graph, and press the Start Test button.

The test will begin if everything is correct in your script. If something is wrong, perhaps a spelling mistake or similar error, then the Ondio window will reopen, automatically highlight the line where the error was made, and explain what is wrong.

Many mistakes that you are likely to encounter have context sensitive help. Press the F1 key for further information about what went wrong.

Like all tests in **NEXYGEN**, it is only possible to run a test in each graph once. Also, it is not possible to change the test setup once it has been started. If something goes wrong in the script after the test has started then the Ondio window will be opened, but the script will be read-only. If you need to make a change to the script you will need to insert a new test in the batch first.

---

### 4.21 PARAMETERS WITH UNITS

It is an easy mistake to forget about the units and square brackets altogether when you are using a function such as **Stage**, which requires them. If you always work in millimetre units, then it is easy to accidentally write:

## 4.0 TUTORIAL

### Stage 10, 2

Instead of

```
Stage [10mm], [2mm/s]
```

Unlike most errors with units, this one will not be highlighted in red as you type it. The error is detected when you run the test, the Ondio window reopens, it highlights the line with the error, and you will see an error message saying:

Failure in Stage (The specified value is the wrong type)

---

### 4.22

### ADDING A RESULT

In this example we will make our first change to the Multi-Stage starting point, by adding an extra result. The Primary Script should start out as:

```
Stage [5mm], [30mm/min]
Stage [10mm], [30mm/min]
Stage [-10mm], [30mm/min]
Stage [10mm], [30mm/min]
Result "Maximum Load", LoadAtMax
Result "Work", Work
```

The script already reports the maximum load during the test. We will extend the script to also report the extension at which this load was reached, using the **ExtnAtMax** built-in property described previously. The final script is:

```
Stage [5mm], [30mm/min]
Stage [10mm], [30mm/min]
Stage [-10mm], [30mm/min]
Stage [10mm], [30mm/min]
Result "Maximum Load", LoadAtMax
Result "Work", Work
Result "E", ExtnAtMax
```

An extra line has been added, again using the **Result** command. The first parameter to this command is the column title. This can be any string, but in this case we have chosen to use an abbreviation "E".

The second parameter is the value of the result. As expected, this is the built-in property **ExtnAtMax**.

### CALCULATIONS

In all the examples so far, all of the values with units have been either:

- Written out in full using square brackets, such as [10mm].
- The value of a built-in property, which comes automatically as a value with units.

It is also possible to calculate using values with units. Ondio has all the usual calculation operators that you might expect from a script language (+ - \* and /), but with the extra power of automatically tracking your units.

Your calculation must be surrounded by square brackets to enable this unit-tracking.

In the script below an extra result has been added for the Stress at the maximum load point. This is calculated by dividing the maximum load (which was in the original example) by the sample cross-section area (which is available as the **Area** built-in property).

```
Stage [5mm], [30mm/min]
Stage [10mm], [30mm/min]
Stage [-10mm], [30mm/min]
Stage [10mm], [30mm/min]
Result "Maximum Load", LoadAtMax
Result "Work", Work
Result "E", ExtnAtMax
Result "Stress", [LoadAtMax/Area]
```

It is essential that the calculation is surrounded by square brackets. Without the square brackets here, the / symbol would mean ordinary division. Ordinary division does not understand values with units, and you will get an error message.

With the square brackets in place around the whole calculation, the / symbol means unit-tracking-division. The result of this division is a value in pressure units, as expected for a stress result. You may remember that when a value with units is written in a script that it will be highlighted in blue. In addition, a units-tracking calculation will be highlighted in purple.

## 4.0 TUTORIAL

Note that Ondio will always calculate the correct value for the stress, whether the operator entered the sample dimensions in inches or millimetres. Once the stress is in the table of results anyone can choose their preferred units (MPa, kgf/cm<sup>2</sup>, lbf/in<sup>2</sup> etc) by right-clicking on the column. You never need to include a scaling factor in the calculation yourself.

---

### 4.24 LOAD LIMITS

The previous example moved the machine to four different positions at the same speed, 30mm/min.

All four positions were extension measurements, measured from the preload point.

```
Stage [5mm], [30mm/min]
Stage [10mm], [30mm/min]
Stage [-10mm], [30mm/min]
Stage [10mm], [30mm/min]
```

It is also possible to specify a load value as the first parameter to the **Stage** command, which causes the machine to drive up to the specified load before continuing. The following script drives up to 1kN, then back to zero. It uses the same **ExtnAtMax** property to measure the extension at maximum load (and the maximum load should be very close to 1kN).

```
Stage [1kN], [30mm/min]
Stage [0mm], [30mm/min]
Result "E", ExtnAtMax
```

---

### 4.25 LOAD RATES

It is also possible for the speed to be a load rate.

```
Stage [1kN], [100N/min]
Stage [0mm], [100N/min]
Result "E", ExtnAtMax
```

## 4.0 TUTORIAL

Load rate stages are not supported on every type of materials testing machine. If you try it on such a machine you will see an error message:

Failure in RunStage (Could not start stage)

---

### 4.26 A CYCLE TEST

The following script will perform a test which involves cycling between 10mm and 0mm three times. It reports three results, for the load at the top of each cycle.

```
Stage [10mm], [1mm/s]  
Result "Load1", Load  
Stage [0mm], [1mm/s]
```

```
Stage [10mm], [1mm/s]  
Result "Load2", Load  
Stage [0mm], [1mm/s]
```

```
Stage [10mm], [1mm/s]  
Result "Load3", Load  
Stage [0mm], [1mm/s]
```

It is easy to create a test like this for a small number of cycles using copy and paste, but the script would get very big if you added more cycles.

---

### 4.27 MANY CYCLES

The following script will perform eight cycles between 20mm and 0mm

```
for i=1 to 8  
  Stage [20mm], [1mm/s]  
  Stage [0mm], [1mm/s]  
next
```

The first line starts a **for** loop, which causes the following statements to be run multiple times. Everything up to the following **next** is inside the loop, and will be repeated as specified.

## 4.0 TUTORIAL

The **for** line controls how many times the loop will be repeated. The variable **i** is used as a loop counter. The loop is run with **i** set to every value between 1 and 8 in turn, a total of eight cycles.

In this example the statements between the **for** and the **next** have been indented. You do not need to do this, but it may make your scripts easier to read.

---

### 4.28 OTHER CYCLE LIMITS

If you do not want to perform a fixed number of cycles then you will need to add an extra line to break out of that loop early. For example you may want to run the test for exactly one minute, but you do not know how many cycles that would take.

The following script makes two changes to the previous example.

Firstly, the cycle limit has been set to a large number, 10000. The script never reaches this limit because another condition has been added to break out of the loop.

```
for i=1 to 10000  
  Stage [20mm], [1mm/s]  
  Stage [0mm], [1mm/s]  
  if [TestTime]>[1min]] then exit for  
next
```

The extra line compares the test time against the limit of one minute. If the test has been running longer than one minute then it uses the **exit for** statement to break out of the loop.

---

### 4.29 USING THE LOOP COUNTER

You can use the loop counter variable in calculations inside the loop.

The original three stage cycle test example created three results for the load at the top of each cycle. Each result needs a different name so that each cycle gets its own

## 4.0 TUTORIAL

column in the table of results. This can be achieved using the loop counter variable.

```
for i=1 to 8
  Stage [20mm], [1mm/s]
  Result "Load"&i, Load
  Stage [0mm], [1mm/s]
next
```

The Result line is very similar to each of the three lines in the previous example. The value of the result is the **Load** built-in property. However in this case the title of the result is a formula.

The result title is calculated by concatenating (joining) two strings together. The first part is the string "Load", which is joined with the loop counter. **&** is the symbol for the concatenation operator.

This produces eight different strings and eight different results, "Load1", "Load2", up to "Load8". The outcome of this script is shown in the results table, Figure 3.

	Load1	Load2	Load3	Load4	Load5	Load6	Load7	Load8
24	36.01 N	35.51 N	35.6 N	35.50 N	35.59 N	35.68 N	35.68 N	35.68 N

Figure 3

If you have a very large number of cycles you may not want to record the peak load at each cycle, perhaps only the first and the last. You can do this by checking the loop counter in an **if** statement.

```
for i=1 to 10
  Stage [10mm], [1mm/s]
  if i=1 then Result "First", Load
  if i=10 then Result "Last", Load
  Stage [0mm], [1mm/s]
next
```

---

### 4.30

### CALCULATING STAGE LIMITS

The previous example showed how a function that is usually used with a constant value (the result title) can also be used with a formula. This is true for all Ondio functions.

## 4.0 TUTORIAL

For example, the following script uses a formula in the stage limit to crush a sample by one quarter of its height.

```
for i=1 to 10
  Stage [Height * 0.25], [1mm/s]
  Stage [0mm], [1mm/s]
next
```

The built-in property **Height** contains the sample height set on the first page of the Ondio setup. This type of test is particularly useful when the height option is set to "Auto-Measure"

---

### 4.31 TEST PROGRESS

If you are writing a test that will run for a long time it is convenient to provide a quick way for the anyone to tell how far the test has run. Using the **Progress** command you can set the message which is displayed both in the status bar of the graph, and on the **NEXYGEN** screen saver.

For example, the following script will display a count of cycles.

```
for i=1 to 100
  Progress "Cycle number " & i
  Stage [20mm], [1mm/min]
  Stage [10mm], [1mm/min]
next
```

---

### 4.32 ADVANCING A TEST

For some tests it is convenient to allow the test operator to control the progress of the test. For example you may usually precondition a sample with three cycles before taking your measurements, but occasionally you want to allow the operator to skip the preconditioning.

**NEXYGEN** allows the test progress to be controlled using two buttons on the graph,



Advance Test, which is usually used for advancing to the next part of a test, and

## 4.0 TUTORIAL



End Test, which is usually used to end a test early.

Your Ondio script has complete control of these buttons. To turn a button on, use the **AdvanceOn** or **EndOn** commands. To turn it off again, use **AdvanceOff** or **EndOff**.

For example the following script will precondition a sample with three cycles and then measure a load. However, it also allows the operator to skip the preconditioning.

```
AdvanceOn
Progress "Preconditioning Sample"
Stage [10mm], [1mm/s]
Stage [0mm], [1mm/s]
Stage [10mm], [1mm/s]
Stage [0mm], [1mm/s]
Stage [10mm], [1mm/s]
Stage [0mm], [1mm/s]
AdvanceOff
Progress "Measuring Load at 10mm"
Stage [10mm], [1mm/s]
Result "Load at 10mm", Load
```

When the advance button is pressed Ondio will stop the current stage. All subsequent Stage commands are effectively ignored until the following **AdvanceOff** command.

Whether he pressed it or not, the final drive to 10mm will happen normally.

N.B. Pressing the Advance button will not necessarily stop the machine. To do this, use the StopMachine function.

---

### 4.33

### ADVANCING A LONG TEST

In the previous test the operator could advance over three preconditioning cycles. If you need more cycles then you would write a loop:

## 4.0 TUTORIAL

```
AdvanceOn
Progress "Preconditioning Sample"
for i=1 to 1000
  Stage [10mm], [1mm/s]
  Stage [0mm], [1mm/s]
next
AdvanceOff
Progress "Measuring Load at 10mm"
Stage [10mm], [1mm/s]
Result "Load at 10mm", Load
```

This script may not have exactly the desired effect if the operator presses the advance button near the start of the test. Ondio will ignore the following 999 cycles, as expected, but it still has to go round the loop 999 times and this make take longer than you want.

This problem can be solved by adding a line to the script to break out of the loop quickly if the built-in **Advancing** property is set.

```
AdvanceOn
Progress "Preconditioning Sample"
for i=1 to 1000
  Stage [10mm], [1mm/s]
  Stage [0mm], [1mm/s]
  if Advancing then exit for
next
AdvanceOff
Progress "Measuring Load at 10mm"
Stage [10mm], [1mm/s]
Result "Load at 10mm", Load
```

---

### 4.34

### THE DIFFERENCE BETWEEN END AND ADVANCE

Conventionally the End Test button is used to skip to the end of a test, and the Advance Test button is used to skip on to the next part. However, you can use either button any way you need, since you have complete control over how much of the test is skipped. Each button has three commands or properties that behave in exactly the same way:

| Advance Button | End Button

## 4.0 TUTORIAL

Enable the button	<b>AdvanceOn</b>	<b>EndOn</b>
Disable the button	<b>AdvanceOff</b>	<b>EndOff</b>
Whether the button has been pressed	<b>Advancing</b>	<b>Ending</b>

### 4.35

### A MODULUS TEST

The following script measures the gradient of the load/extension trace between two fixed extension points. This is the sample modulus.

This script uses two stages, driving to each of the two limits where measurements must be taken. The modulus is calculated as the difference in load between the end of the two stages, divided by the difference in extension.

The difference in extension is constant, because both stages have extension limits. The difference in load is measured.

```
Stage [15mm], [30mm/min]
set L1=Load
Stage [20mm], [30mm/min]
Result "Modulus", [(Load-L1)/[5mm]]
```

This script uses the **set** statement to create a variable **l1** holding the load at end of the first stage.

At the end of the second stage, the script can calculate the change in load by subtracting the variable **l1** from the current **Load**.

Note: that you always need to use the **set** command when creating or changing a variable that contains a value with units (such as **set L1=Load**). You must not use **set** to create or change a variable containing a number, string, or any other value (everything except an advanced data type called an object, which most Ondio scripts never use).

This script will produce the trace shown in Figure 4.

## 4.0 TUTORIAL

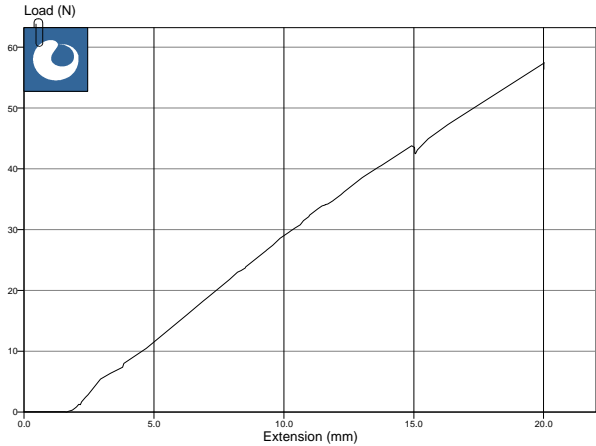


Figure 4

### 4.36

### A MORE ACCURATE MODULUS TEST

Figure 4 was produced using a rubber sample. You can see that the load drops briefly at the 15mm point. This happens because the machine pauses, and the sample relaxes. You can see this effect clearly in Figure 5 where the axes have been changed to extension against time. The extension pauses at 15mm for almost one second.

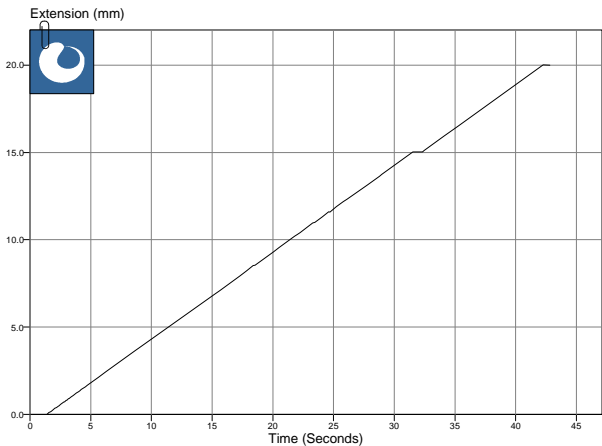


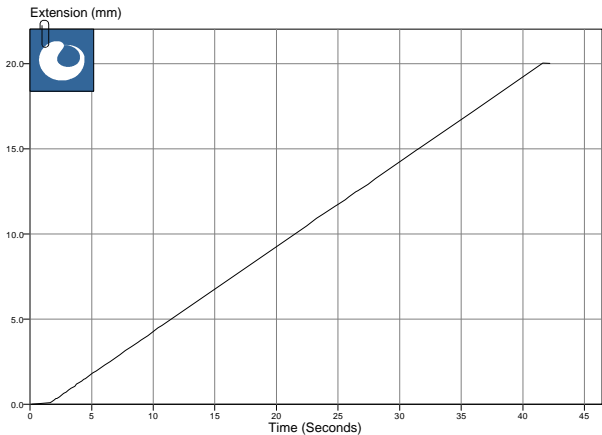
Figure 5

## 4.0 TUTORIAL

This pause is not a problem for other tests, or for materials that do not relax such as metals. However it can be avoided by writing the script slightly differently. The following script only uses one stage, and uses the function **StaticLoad** to examine the graph trace after the test has finished.

```
Stage [20mm], [30mm/min]
set L1=StaticLoad([15mm])
Result "Modulus", [(Load-L1)/[5mm]]
```

With this approach the test is more accurate because the machine does not need to pause. In Figure 6 you can see that the extension rises continuously during the test.



**Figure 6**

This test can be improved further by using the **StaticLoad** function to measure the load at the 20mm point too. This allows the stage limit to be increased a little, so that the machine is moving smoothly through the point where measurements will be taken.

```
Stage [21mm], [30mm/min]
set L1=StaticLoad([15mm])
set L2=StaticLoad([20mm])
Result "Modulus", [(L2-L1)/[5mm]]
```

---

### 4.37 THE STATIC LOAD FUNCTION

The **StaticLoad** function takes one parameter, an extension value that should be looked up on the graph. It returns the load at that point. If the trace had reached that extension more than once, such as in a cycle test, it will return the load for the first time it was reached.

A complementary function **StaticExtn** takes a single load parameter, and returns the extension when that load was first reached.

---

### 4.38 A LIMITATION OF CYCLE TESTS

Some built-in functions examine the whole test. For example **LoadAtMax** returns the maximum load over the whole test so far.

In a complex test you may need access to the same information, but just over one cycle or one stage.

Most of the whole-test functions have single-stage equivalents. For example, the maximum load over the previous stage is available in the property **LoadAtStageMax**.

For a full list of equivalents, see the reference section.

Combining two previous examples, the following test will precondition a sample with three cycles, then calculate a modulus. The **StaticLoad** function has been substituted with **StageStaticLoad**, since **StaticLoad** would return the load on the first cycle which is actually part of the preconditioning.

```
for i=1 to 3
  Stage [20mm], [30mm/min]
  Stage [0mm], [30mm/min]
next
Stage [21mm], [30mm/min]
set L1=StageStaticLoad([15mm])
set L2=StageStaticLoad([20mm])
Result "Modulus", [(L2-L1)/[5mm]]
```

### ANOTHER WAY TO BEAT PAUSES

The pauses in Figure 5 were caused because the testing machine did not know what it was going to do next once it had finished each stage. The machine paused for as long as it took the machine to inform the script that one stage had finished, the script to store some measurements in some variables, and the script to use the **stage** command again to set the machine in motion.

For a Modulus test, the best solution is to use just one stage and the **StaticLoad** function. However for some other tests you need to run multiple stages without pauses. This can be necessary if the speed or direction of movement must change, or if you need to perform some other measurement between stages (such as reading **LoadAtStageMax**, or communicating with another program).

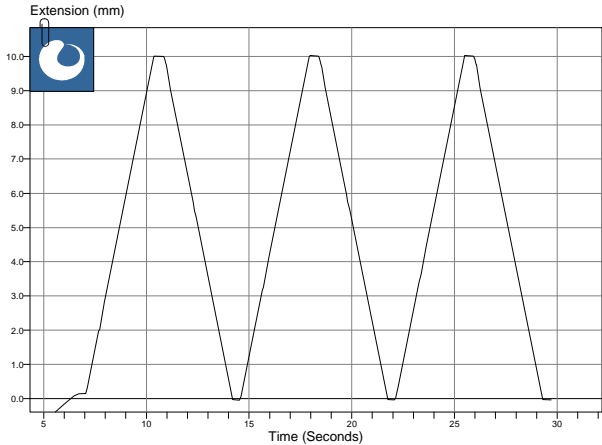
It is possible to see the same effect in Figure 7, a graph from the script below that was first shown on page 31.

```
Stage [10mm], [3mm/s]  
Result "Load1", Load  
Stage [0mm], [3mm/s]
```

```
Stage [10mm], [3mm/s]  
Result "Load2", Load  
Stage [0mm], [3mm/s]
```

```
Stage [10mm], [3mm/s]  
Result "Load3", Load  
Stage [0mm], [3mm/s]
```

## 4.0 TUTORIAL



**Figure 7**

Since the problem is caused by the machine not knowing what it should be doing from one stage to the next, it is possible to solve the problem by giving the machine the stage details in advance. To achieve this, the script must replace the usual **Stage** function with a combination of two others. These two functions are usually **SetupStage** and **RunStage**.

The first of these functions, **SetupStage**, is used to send stage information down to the machine. Unlike **Stage**, it does not start the machine moving. The machine stores the stages, building up a plan of what the test will do.

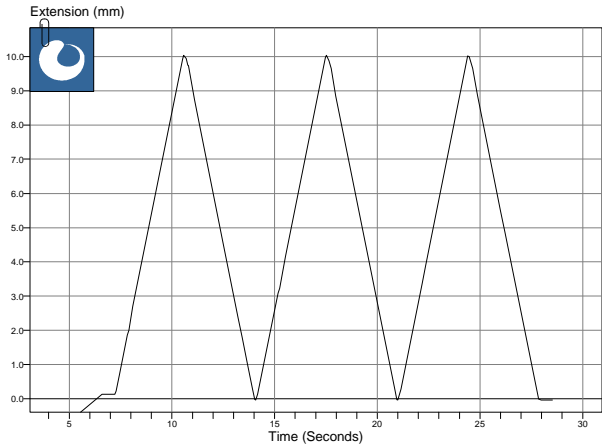
The second function, **RunStage**, instructs the machine to carry out one step of the plan. Like **Stage** it will return when the first stage limit is reached, and the script can use properties such as **Load** to calculate some results from that one stage. However, unlike when using **Stage** the machine already knows what it should be doing next. It can start on the next stage with a minimal pause.

Each call to **SetupStage** must be matched with a call to **RunStage** later in the script.

The modified script is shown below, and the resulting graph in Figure 8.

## 4.0 TUTORIAL

```
SetupStage [10mm], [3mm/s]
SetupStage [0mm], [3mm/s]
SetupStage [10mm], [3mm/s]
SetupStage [0mm], [3mm/s]
SetupStage [10mm], [3mm/s]
SetupStage [0mm], [3mm/s]
RunStage
Result "Load1", Load
RunStage
RunStage
Result "Load2", Load
RunStage
RunStage
Result "Load3", Load
RunStage
```



**Figure 8**

The unwanted pauses of Figure 7 are avoided because the machine can start on the next step of the plan. It is important that your script keeps up with the plan too, by calling **RunStage** at the appropriate time. If the machine gets too far ahead, because your script is slow in calling **RunStage**, then it will have to pause anyway to allow the script to catch up.

### 4.40

### THE RUNALLSTAGES FUNCTION

Every call to **SetupStage** must be followed by a matching **RunStage** later in the script.

If a test involves a large number of stages but does not need measurements taken between stages then it is more convenient to use the **RunAllStages** function.

For example, the following script uses over 60 stages to move the machine in a sine wave. The loop sends all of stages to the machine, and **RunAllStages** is used to run them all.

```
for i =0 to 2*3.14 step 0.1
  set l=[[100mm]*sin(i)]
  set s=[[10mm/s]*abs(cos(i))]
  SetupStage l,s
next
RunAllStages
```

---

### 4.41

### THE SECONDARY SCRIPT

All of the scripts so far have been *Primary Scripts*. They control what happens *during* the test.

Some types of test also require a *Secondary Script*, which controls what happens *after* the test. There are several reasons that you might need to use a secondary script, which are explained in the following sections.

There are also some things that you can not do in a secondary script:

- You can not control the machine, using **Stage** or similar commands.
- You can not use the built-in properties that relate to the previous stage, such as **Load** or **LoadAtStageMax**. However you can use whole-test properties such as **LoadAtMax**.

### MARKERS

Ondio lets you place markers on the Graph using the MarkerNow function. Figure 9 shows a tearing test with three markers.

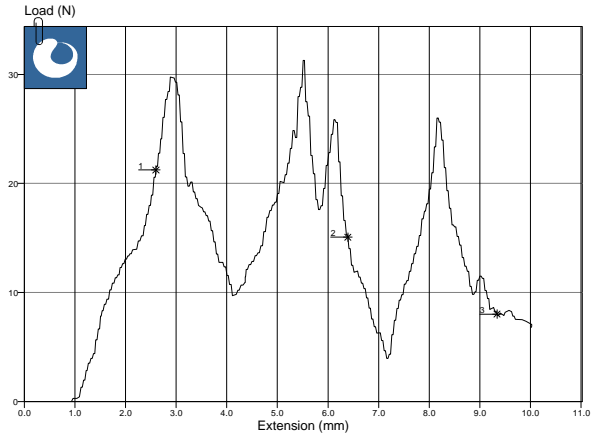


Figure 9

An extra function is available in the secondary script, **MarkerLoad**, which returns the load value at the numbered marker point. For example the following secondary script will produce the results table shown in Figure 10.

```
Result "M1", MarkerLoad("1")
Result "M2", MarkerLoad("2")
Result "M3", MarkerLoad("3")
```

	M1	M2	M3
13	21.23 N	15.07 N	8.026 N

Figure 10

The intention in this test is that the operator moves the markers onto the three highest peaks, using the move marker tool on the Graph toolbar



Figure 11 shows the graph after this has been done, and Figure 12 shows an updated results table.

## 4.0 TUTORIAL

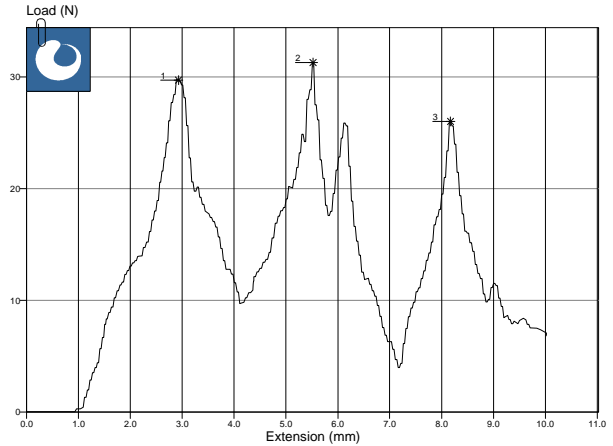


Figure 11

	M1	M2	M3
13	29.71 N	31.27 N	26.00 N

Figure 12

Note that the results were changed between Figure 10 and Figure 12. This update happened because Ondio automatically re-runs the secondary script every time a Marker is moved.

This explains why you can not use the **MarkerLoad** function in the primary script. The primary script can only be run once.

---

### 4.43

### TEST-GLOBAL VARIABLES

In some tests you may need to perform a calculation involving some values that are only available in the primary script and others that are only available in the secondary script.

The **Global** function makes it possible to transfer values from the primary script into the secondary script. For further information, see the Reference section of this manual on page 74.

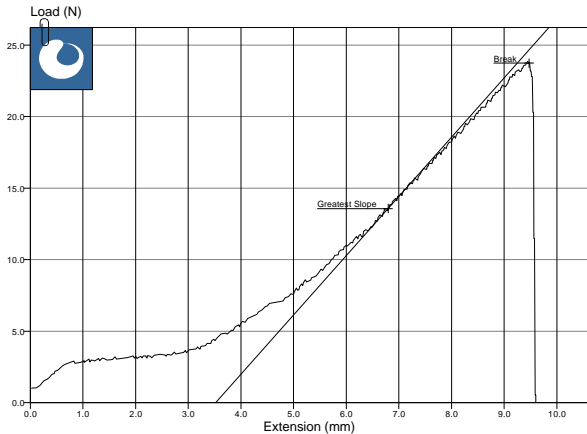
### 4.44

### AN AUTOMATIC MODULUS TEST

The modulus test on page 39 can be simplified even further using the secondary script.

The secondary script has a built-in property **Modulus** which for most tests will automatically determine the steepest gradient.

Using this property has some other advantages; it will mark the steepest slope point on the trace, and draw a tangential line along the trace.



**Figure 13**

This is achieved using the one line secondary script:

```
Result "Modulus", Modulus
```

### 4.45

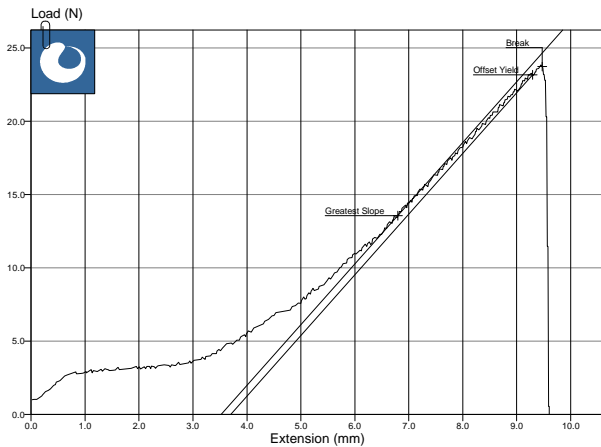
### AUTOMATIC OFFSET YIELD

Ondio provides several other convenient auto-calculated values in the primary script, such as measurements taken; at the break point and at the offset yield point (also known as Proof Stress).

The secondary script has two built-in properties **LoadAtOffsetYield** and **ExtnAtOffsetYield**. As in the modulus case, Ondio will automatically mark the offset yield construction lines on the graph, and it place an

## 4.0 TUTORIAL

event marker at the offset yield point, as shown in Figure 14.



**Figure 14**

The secondary script must specify the extension offset before using either of these properties, using the **SetOffsetExtn** command. Typically the extension is specified as a fraction of the sample gauge length. For example, the following secondary script will calculate a 0.5% proof stress value.

```
SetOffsetExtn([Height*0.005])  
Result "Proof Stress", LoadAtOffsetYield
```

---

### 4.46

### BREAK RESULTS

It is possible to set up Ondio to automatically react when the sample breaks. Different samples have different break characteristics, which can be specified on Ondio's first page. For more information see page 22.

The default reaction when the break detector triggers is to end the current stage and skip over all subsequent stages until the test is over. This behavior is exactly the same as when the Advanced button is pressed, described in detail on page 34.

## 4.0 TUTORIAL

If the break detector triggered three extra properties are available in the secondary script; **LoadAtBreak**, **ExtnAtBreak**, and **WorkToBreak**.

Note that these properties are only available if the sample broke. The secondary script must check this first, as shown in the following secondary script example.

```
if Broken then
  Result "Load", LoadAtBreak
  Result "Work", WorkToBreak
end if
```

---

### 4.47

### USING THE BREAK DETECTOR IN A CYCLE TEST

All of Ondio's break detectors involve looking for a drop in load. Clearly there will be some complications if you want to cycle until break since the load is *supposed* to drop during one half of a cycle test.

For tests involving large numbers of fast cycles, it may be more important to know on which cycle the sample broke, rather than the exact load and extension during that last cycle. In this case it is easier to write the end condition in the primary script yourself, rather than using the built in break detector. The following script shows an example.

```
for i=1 to 10000
  Stage [10mm], [10mm/s]
  if [Load<[10N]] then exit for
  Stage [0mm], [10mm/s]
next
```

If your cycles are slow and do not involve a large drop in load then the built-in break detector can still be used. If you are using a large number of cycles then you may need to use the technique shown on page 36 to break out of the loop quickly, substituting the **Advancing** property with **Broken**.

---

### 5.1 INTRODUCTION

Ondio supports the following functionality in respect of data acquisition:

- Continuous acquisition of data on an analogue input channel.
- Termination of a drive stage when an analogue input channel falls above or below a defined threshold.
- Automatic conversion of the input voltage to the required unit (e.g. an extensometer reading would be converted to the appropriate extension)
- Termination of a drive stage when a digital input line becomes high/low.
- Change the state of a digital output line (e.g. to switch on/off some external hardware)

The data acquisition cards supported at this point are the National Instruments PC-516 (for 16 bit precision), and the PC-LPM-16PnP (for 12-bit precision). There is a possibility of using other cards not specifically catered for by Ondio, but their successful use cannot be guaranteed, and no support will be provided in this respect.

The required driver for data acquisition is the NI-DAQ 6.6.0 or later, as provided with the DAQ card or available from the National Instruments web-site at <http://www.ni.com>.

The following sections assume that the appropriate NI DAQ card and driver has been successfully installed and configured.

---

### 5.2 DEVICE SETTINGS

Data acquisition card settings are test-specific. On the Ondio Edit menu, use the **Device Settings** screen to configure the card:

## 5.0 DATA ACQUISITION

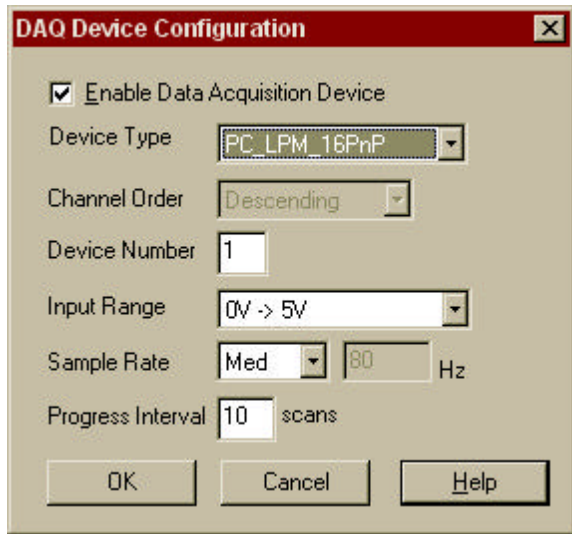


Figure 15

If data acquisition is not being used in the test, the checkbox should be unchecked (the default).

Select the Device Type according to the DAQ card being used.

Channel Order is not relevant unless the DAQ card is an unsupported one. It defines the order the analogue input channels are configured in. i.e. channel 0 first (Ascending) or channel 0 last (Descending). Different cards may have different requirements in this respect.

The Device Number should correspond with the device number recognised by the NI-DAQ driver, for that card. If only one card is installed, it will usually have device number 1.

Input Range defines the expected analogue input voltage range for the device. The choice available depends on the selected card.

Sample Rate may be Low, Med or High depending on the type and length of test being performed.

The Progress Interval specifies the interval in scans between progress events. This value should only be altered under

## 5.0 DATA ACQUISITION

exceptional circumstances, such as a problem synchronising the readings from the card.

### 5.3

### ANALOGUE INPUTS

The analogue input channels used for the test must be configured on the **Analogue Inputs** screen, accessed from the Ondio Edit menu:

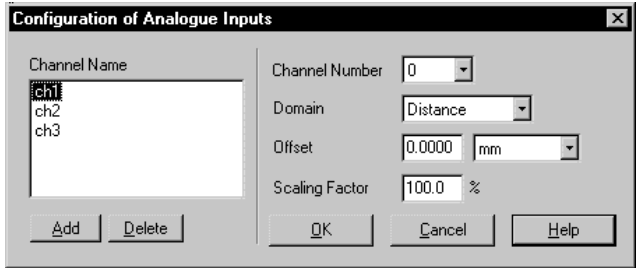


Figure 16

Channel Name lists all the currently configured channels. Click Add to create a new one. Click Delete to remove the selected one. Click right mouse button Rename, to change the name of the selected channel. New channels default to names such as 'New Channel', 'New Channel1' etc. Select a channel name to view its details on the right hand side of the dialogue.

Channel Number is the number of the analogue input channel on the card. In order to successfully configure the card, the channels must be used consecutively, i.e. 0 then 1, then 2 etc, depending on the number of input channels being used. Therefore, the first channel in the list should be assigned to channel number 0, then next to channel number 1, and so on.

Domain determines the type of quantity being measured. i.e. the quantity the input voltage should be converted to on the axis. E.g. If an extensometer is providing the input to the DAQ card, then the domain will be Distance.

## 5.0 DATA ACQUISITION

Offset and Scaling Factor tell NEXYGEN how to convert the input voltage into the required unit on the axis.

E.g.

1. An extensometer is providing the input to the DAQ card. The card is configured for a input range of 0-5V (See Device Settings). 0V corresponds to 0mm deflection and 5V corresponds to a 10mm deflection. Thus, the Offset will be 0mm and the Scaling Factor will be 200%.

2. A temperature gauge is providing the input to the DAQ card. The card is configured for a input range of 0-10V (See Device Settings). 0V corresponds to -2°C and 10V corresponds to +2°C. Thus, the Offset will be -2°C and the Scaling Factor will be 40%.

---

### 5.4

### EXAMPLES

Drive until the extensometer on analogue input channel 1 falls below 2mm, or until the cross-head has driven to 10cm. Finally, set digital input line 0 on port 0 low.

```
AnalogueBelowOn "ch1", [2mm]
```

```
Stage [10cm], [1cm/s]
```

```
AnalogueBelowOff "ch1"
```

```
SetDigitalLow 0,0
```

Drive until digital input 0 on port 0 goes high, or until the cross-head has drive 10cm.

```
DigitalHighOn 0,0
```

```
Stage [10cm], [1cm/s]
```

```
DigitalHighOff 0,0
```

---

<b>6.1</b>	<b>PRIMARY SCRIPT PROPERTIES</b>	
	<b>Load</b>	Current load
	<b>Extn</b>	Current extension
	<b>Work</b>	† Work since start of test
	<b>TestTime</b>	Time since start of test
	<b>Height</b>	† Sample dimension, if appropriate
	<b>Area</b>	†
	<b>Width</b>	†
	<b>Breadth</b>	†
	<b>Diameter</b>	†
	<b>LoadAtMax</b>	† Maximum load so far
	<b>ExtnAtMax</b>	† Extension at maximum load
	<b>LoadAtMin</b>	† Minimum load so far
	<b>ExtnAtMin</b>	† Extension at minimum load
	<b>StageNumber</b>	Number of stages run
	<b>Broken</b>	† Whether the sample is broken
	<b>Advancing</b>	Whether the advance button has been pressed
	<b>Ending</b>	Whether the end test button has been pressed
	<b>LoadAtStageMax</b>	Maximum load over the previous stage
	<b>ExtnAtStageMax</b>	Extension at that point
	<b>LoadAtStageMin</b>	Smallest load over the previous stage
	<b>ExtnAtStageMin</b>	Extension at that point

† Also available in the Secondary script

### 6.2

### PRIMARY SCRIPT FUNCTIONS

<b>Stage</b>	Run a stage
<b>StageHold</b>	Run a stage, which involves a hold period
<b>SetupStage</b>	Send a stage to the machine, but do not run it
<b>SetupStageHold</b>	Send a stage which involves a hold time to the machine, but do not run it
<b>RunStage</b>	Run a previously sent stage
<b>RunAllStages</b>	Run all previously sent stages
<b>BreakOff</b>	Inhibit the break detector
<b>BreakOn</b>	Re-enable the break detector
<b>AdvanceOn</b>	Enable the Advance button
<b>AdvanceOff</b>	Disable the Advance button
<b>EndOn</b>	Enable the End Test button
<b>EndOff</b>	Disable the End Test button
<b>TearOn</b>	Switch in to tear-test mode
<b>TearOff</b>	Switch out of tear-test mode
<b>StaticLoad</b>	† Look up the load at a specific extension
<b>StaticExtn</b>	† Look up the extension at a specific load
<b>StageStaticLoad</b>	Look up the load at a specific extension over the previous stage
<b>StageStaticExtn</b>	Look up the extension at a specific load over the previous stage
<b>Result</b>	† Set a result
<b>Global</b>	Set a test-global variable
<b>Progress</b>	† Set the graph status-bar progress message
<b>GetExtraResult</b>	† Get user-defined extra result (for use in calculations)
<b>ExtraLoadAxis</b>	Add a user-defined load axis
<b>ExtraExtnAxis</b>	Add a user-defined extension axis
<b>MarkerNow</b>	† Create a marker with the specified name
<b>EventNow</b>	Create a new event

## 6.0 REFERENCE

<b>UseSharpDropDetector</b>	Switch to sharp drop break detector
<b>UsePercentageDropDetector</b>	Switch to percentage drop break detector
<b>Pause</b>	Pause for a specified period
<b>AnalogueAboveOn</b>	Enable the analogue input above detector for the specified channel
<b>AnalogueAboveOff</b>	Disable the analogue input above detector for the specified channel
<b>AnalogueBelowOn</b>	Enable the analogue input below detector for the specified channel
<b>AnalogueBelowOff</b>	Disable the analogue input below detector for the specified channel
<b>DigitalLowOn</b>	Enable the digital input low detector for the specified line
<b>DigitalLowOff</b>	Disable the digital input low detector for the specified line
<b>DigitalHighOn</b>	Enable the digital input high detector for the specified line
<b>DigitalHighOff</b>	Disable the digital input high detector for the specified line
<b>ValueOfInput</b>	Return the value of the specified analogue input channel
<b>SetDigitalHigh</b>	Set the specified digital output line high
<b>SetDigitalLow</b>	Set the specified digital output line low

† Also available in the Secondary script

---

### 6.3 SECONDARY SCRIPT PROPERTIES

<b>LoadAtBreak</b>	Load at the break point
<b>ExtnAtBreak</b>	Extension at the break point
<b>WorkToBreak</b>	Work done up to the break point
<b>Modulus</b>	Automatic tangential modulus the Load/Extension graph
<b>LoadAtOffsetYield</b>	Load at automatic offset yield
<b>ExtnAtOffsetYield</b>	Extension at that point
<b>LoadAtGreatestSlope</b>	Return load at point of greatest slope
<b>ExtnAtGreatestSlope</b>	Return extension at point of greatest slope.

The secondary script may also use some of the Primary Script properties.

---

### 6.4 SECONDARY SCRIPT FUNCTIONS

<b>Global</b>	Get a test-global variable
<b>MarkerLoad</b>	The load at a marker point
<b>MarkerExtn</b>	The extension at a marker point
<b>MarkerWork</b>	Work done up to a marker point
<b>MarkerTime</b>	The time at a marker point
<b>MarkerInput</b>	The value of the specified analogue input channel at a marker point
<b>SetOffsetExtn</b>	Set the extension used by the automatic offset yield
<b>StaticWork</b>	Return work done up to a specified load or extension
<b>InputAtMax</b>	The value of the specified analogue input channel at point of maximum load
<b>InputAtMin</b>	The value of the specified analogue input channel at point of minimum load

## 6.0 REFERENCE

<b>InputAtStageMax</b>	The value of the specified analogue input channel at point of maximum load for the stage
<b>InputAtStageMin</b>	The value of the specified analogue input channel at point of minimum load for the stage

The secondary script may also use some of the Primary Script functions.

### A TABLE OF UNITS

---

#### DISPLACEMENT UNITS

m  
mm  
cm  
in  
 $\mu\text{m}$   
microns  
thous

#### FORCE UNITS

N  
kN  
kgf  
lbf  
gf

#### AREA UNITS

$\text{m}^2$   
 $\text{mm}^2$   
 $\text{cm}^2$   
 $\text{in}^2$

#### VOLUME UNITS

$\text{mm}^3$   
 $\text{cm}^3$   
 $\text{in}^3$

#### TIME UNITS

s  
ms  
min  
hours  
days

#### SPEED UNITS

m/s  
m/min

## 6.0 REFERENCE

m/hr  
cm/min  
mm/s  
mm/min  
mm/hr  
in/s  
in/min  
in/hr

### FORCE RATE UNITS

N/s  
N/min  
N/hr  
kN/s  
kN/min  
kN/hr  
kgf/s  
kgf/min  
kgf/hr  
lbf/s  
lbf/min  
lbf/hr

### STIFFNESS UNITS

N/m  
N/mm  
kN/m  
lbf/in  
kgf/mm

### PRESSURE UNITS

MPa  
kPa  
N/m<sup>2</sup>  
N/mm<sup>2</sup>  
Pa  
lbf/in<sup>2</sup>

### STRESS RATE UNITS

N/m<sup>2</sup>s  
MPa/s  
N/mm<sup>2</sup>s

## **6.0 REFERENCE**

### **WORK UNITS**

Nm  
Nmm  
J  
kJ  
kgf.mm

### **MASS UNITS**

kg  
g  
lb  
t

### **DENSITY UNITS**

$\text{g/m}^3$   
 $\text{g/mm}^3$   
 $\text{g/cm}^3$

### **POTENTIAL DIFFERENCE UNITS**

pV  
nV  
 $\mu\text{V}$   
mV  
V  
kV

### **CURRENT UNITS**

pA  
nA  
 $\mu\text{A}$   
mA  
A

### **RESISTANCE UNITS**

Ohms  
kOhms  
MOhms

### **CAPACITANCE UNITS**

pF  
nF

## **6.0 REFERENCE**

$\mu\text{F}$   
mF  
F

### **INDUCTANCE UNITS**

H

### **CHARGE UNITS**

C

### **FREQUENCY UNITS**

Hz  
kHz  
MHz

---

### 6.6 ALPHABETICAL REFERENCE

This section lists all of Ondio's built-in properties and functions, in alphabetic order.

---

Function  
Primary script: Yes  
Secondary: No

#### **AdvanceOff**

This function is used to disable the Advance Test command button, after it has been enabled using the **AdvanceOn** function.

---

Function  
Primary script: Yes  
Secondary: No

#### **AdvanceOn**



This function is used to enable the Advance Test command button in the graph. It can be disabled again using the **AdvanceOff** function.

When the Advance Test command button is pressed Ondio will interrupt the current function, whether **Stage**, **RunStage** or **RunAllStages**. All subsequent calls to those functions will be ignored until either **AdvanceOn** or **AdvanceOff** is called again.

This function is useful to allow the test operator to skip a section of the test. Use **AdvanceOn** at start of the section that he can skip; the button will be available after this point. Use **AdvanceOff** at the end of the section he can skip. If the button is pressed, then any remaining stages up to that point will be skipped. Also the button will be disabled, whether it has been pressed or not.

The following test performs three cycles of preconditioning, then measures the work over a fourth cycle. At any time the preconditioning can be skipped.

## 6.0 REFERENCE

```
AdvanceOn
Stage [10mm], [2mm/s]
Stage [0mm], [2mm/s]
Stage [10mm], [2mm/s]
Stage [0mm], [2mm/s]
Stage [10mm], [2mm/s]
AdvanceOff
Stage [0mm], [2mm/s]
set w=Work
Stage [10mm], [2mm/s]
Stage [0mm], [2mm/s]
Result "Work", [Work-w]
```

After advancing a stage the stage properties such as **Load**, **Extn**, **LoadAtStageMax**, etc, are all set to the values at the point the advance button was pressed. They are not modified by subsequent stages that are skipped.

Although the **stage** function is interrupted, pressing the advance button does not actually stop the machine. It will continue along the path of the interrupted stage until given a different stage to run.

**Note:** Pressing the Advance Test button causes subsequent stages to be skipped until the button is reset, however it does not cause any other statements in the Primary Script to be skipped.

If it is desired to definitely stop the machine when the Advance button is pressed, the following lines should be added to the script:

```
If Advancing then
    StopMachine
end
```

---

Read-only Boolean  
Primary script: Yes  
Secondary: No

### Advancing

This property is true if the Advance Test command button on the graph has been pressed. This button is enabled by the **AdvanceOn** function.

## 6.0 REFERENCE

---

Function  
Primary script: Yes  
Secondary: No

### **AnalogueAboveOff c**

This function requires a Data Acquisition Card, and turns off the analogue above input stage terminator on channel c. It does not affect other channels. For example:

```
AnalogueAboveOn "ch1", [2V]
```

```
Stage [10mm], [2mm/s]
```

```
AnalogueAboveOff "ch1"
```

```
Stage [0mm], [3mm/s]
```

See also **AnalogueAboveOn**, **AnalogueBelowOn**, **AnalogueBelowOff**.

---

Function  
Primary script: Yes  
Secondary: No

### **AnalogueAboveOn c,t**

This function requires a Data Acquisition Card, and is used to enable the analogue input detector on channel name c. The drive stage will be terminated when the analogue input rises above threshold t. For example:

```
AnalogueAboveOn "ch1", [2V]
```

```
Stage [10mm], [2mm/s]
```

Note that it is possible to have two analogue input above detectors active concurrently on different channels, or analogue above and below detectors active on the same channel.

See also **AnalogueAboveOff**, **AnalogueBelowOn**, **AnalogueBelowOff**.

---

Function  
Primary script: Yes  
Secondary: No

### **AnalogueBelowOff c**

This function requires a Data Acquisition Card, and turns off the analogue below input stage terminator on channel c. It does not affect other channels. For example:

```
AnalogueBelowOn "ch1", [2V]
```

```
Stage [10mm], [2mm/s]
```

## 6.0 REFERENCE

**AnalogueBelowOff "ch1"**

**Stage [0mm], [3mm/s]**

See also **AnalogueAboveOn**, **AnalogueAboveOff**, **AnalogueBelowOn**.

---

Function

Primary script: Yes

Secondary: No

### **AnalogueBelowOn c,t**

This function requires a Data Acquisition Card, and is used to enable the analogue input detector on channel name c. The drive stage will be terminated when the analogue input falls below threshold t. For example:

**AnalogueBelowOn "ch1", [2V]**

**Stage [10mm], [2mm/s]**

Note that it is possible to have two analogue input above detectors active concurrently on different channels, or analogue above and below detectors active on the same channel.

See also **AnalogueAboveOn**, **AnalogueAboveOff**, **AnalogueBelowOff**.

---

Read-only property

Primary script: Yes

Secondary: Yes

### **Area**

This property stores the cross sectional area of the sample, provided the sample area option on the first page of the Ondio setup is set to anything other than "Not Applicable".

This property is also useful for reporting results in stress units, for example:

**Result "Max Stress", [LoadAtMax/Area]**

---

Read-only property

Primary script: Yes

Secondary: Yes

### **Breadth**

This property stores the breadth of the sample, if that option is chosen in the first page of Ondio's setup.

## 6.0 REFERENCE

In most cases, the **Area** property is more convenient.

---

Function  
Primary script: Yes  
Secondary: No

### BreakOff

This function is used to inhibit the break detector set up in the first setup page. If the option "Break: Not Expected" has been chosen then this function will have no effect.

---

Function  
Primary script: Yes  
Secondary: No

### BreakOn

This function is used to reset the break detector, either after it has triggered or after it has been inhibited using the **BreakOff** function.

This function resets the break detector using the information in the first setup page. If the option "Break: Not Expected" has been chosen then this function will have no effect.

---

Read-only Boolean  
Primary script: Yes  
Secondary: Yes

### Broken

This property is true if the sample is broken.

It is essential to check this property in the secondary script before using the **LoadAtBreak**, **ExtnAtBreak**, or **WorkToBreak** properties since it is an error to use them if the sample did not break.

It is also necessary to use this property if you are combining a cycle test with a break detector, to ensure that the script breaks out of the cycle if the sample breaks. For example:

```
while [TestTime<[30min]] and not Broken
  Stage [49kN], [1mm/min]
  Stage [50kN], [1mm/min]
wend
```

---

Read-only property  
Primary script: Yes  
Secondary: Yes

### Diameter

## 6.0 REFERENCE

This property stores the diameter of the sample, if that option is chosen in the first page of Ondio's setup.

In most cases, the **Area** property is more convenient.

---

Function  
Primary script: Yes  
Secondary: No

### **DigitalHighOff *p,c***

This function requires a Data Acquisition Card, and is used to disable the digital input detector on port *p*, line *c*.

For example:

```
DigitalHighOn 1,0  
Stage [10mm], [2mm/s]  
DigitalHighOff 1,0  
Stage [0mm], [3mm/s]
```

See also **DigitalLowOff**, **DigitalHighOn**, **DigitalHighOn**.

---

Function  
Primary script: Yes  
Secondary: No

### **DigitalHighOn *p,c***

This function requires a Data Acquisition Card, and is used to enable the digital input detector on port *p*, line *c*. The drive stage will be terminated when the digital line becomes high. For example:

```
DigitalHighOn 1,0  
Stage [10mm], [2mm/s]
```

Make sure the selected port supports digital inputs.

Note that it is possible to have DigitalHigh and DigitalLow detectors active during a stage, provided that there are operating on different lines.

See also **DigitalLowOff**, **DigitalLowOn**, **DigitalHighOff**.

---

Function  
Primary script: Yes  
Secondary: No

### **DigitalLowOff *p,c***

## 6.0 REFERENCE

This function requires a Data Acquisition Card, and is used to disable the digital input detector on port p, line c.

For example:

```
DigitalLowOn 1,0
```

```
Stage [10mm], [2mm/s]
```

```
DigitalLowOff 1,0
```

```
Stage [0mm], [3mm/s]
```

See also **DigitalLowOn**, **DigitalHighOn**, **DigitalHighOff**.

---

Function  
Primary script: Yes  
Secondary: No

### **DigitalLowOn p,c**

This function requires a Data Acquisition Card, and is used to enable the digital input detector on port p, line c. The drive stage will be terminated when the digital line becomes low. For example:

```
DigitalLowOn 1,0
```

```
Stage [10mm], [2mm/s]
```

Make sure the selected port supports digital inputs.

Note that it is possible to have DigitalHigh and DigitalLow detectors active during a stage, provided that there are operating on different lines.

See also **DigitalLowOff**, **DigitalHighOn**, **DigitalHighOff**.

---

Read-only Boolean  
Primary script: Yes  
Secondary: No

### **Ending**

This property is true if the End Test command button on the graph has been pressed. This button is enabled by the **EndOn** function.

---

Function  
Primary script: Yes  
Secondary: No

### **EndOff**

## 6.0 REFERENCE

This function is used to disable the End Test command button, after it has been enabled using the **EndOn** function.

---

Function  
Primary script: Yes  
Secondary: No

### EndOn



This function is used to enable the End Test command button in the graph. It can be disabled again using the **EndOff** function.

The behavior of this button is identical to that of the Advance Test button, which is controlled using the **AdvanceOn** function.

Conventionally the End Test button is used to skip to the end of the test, while the Advance Test button is used to skip to the next part. There are no further differences.

---

Function  
Primary script: Yes  
Secondary: No

### EventNow n

This function is used to add an Event to the graph during the test. The events have user-defined names. For example:

```
Stage [10mm], [2mm/s]
```

```
EventNow "Stage1End"
```

```
Stage [0mm], [2mm/s]
```

```
EventNow "Stage2End"
```

---

Read-only property  
Primary script: Yes  
Secondary: No

### Extn

This property stores the current extension.

See also **Load**, **Work**, and **TestTime**.

More precisely, it stores the extension at the end of the previous stage, therefore it does not change until you run another stage.

---

Read-only property  
Primary script: No  
Secondary: Yes

### ExtnAtBreak

## 6.0 REFERENCE

This property stores the extension at the break point. It is only available if the sample did break, therefore you must check the **Broken** property first.

See also **LoadAtBreak**, **WorkToBreak**

This property may not hold accurate results if you have used the **BreakOn** or **BreakOff** commands in the primary script.

---

Read-only property  
Primary script: No  
Secondary: Yes

### ExtnAtGreatestSlope

This property contains the Load at the Modulus.

See also **LoadAtGreatestSlope**.

---

Read-only property  
Primary script: Yes  
Secondary: Yes

### ExtnAtMax

This property stores the extension at the point where the load reached its maximum over the whole test. If the property is used before the test has finished, it will contain the extension at the maximum load so far.

See also **LoadAtMax**, **ExtnAtStageMax**

---

Read-only property  
Primary script: Yes  
Secondary: Yes

### ExtnAtMin

This property stores the extension at the point where the load reached its minimum over the whole test. If the property is used before the test has finished, it will contain the extension at the minimum load so far.

See also **LoadAtMin**, **ExtnAtStageMin**

---

Read-only property  
Primary script: No  
Secondary: Yes

### ExtnAtOffsetYield

## 6.0 REFERENCE

This property stores the extension at the offset yield point. For details about the offset yield point, see the **SetOffsetExtn** function.

See also **LoadAtOffsetYield**.

---

Read-only property  
Primary script: Yes  
Secondary: No

### ExtnAtStageMax

This property stores the extension at the point where the load reached its maximum over the previous stage.

See also **LoadAtStageMax**.

---

Read-only property  
Primary script: Yes  
Secondary: No

### ExtnAtStageMin

This property stores the extension at the point where the load reached its minimum over the previous stage.

See also **LoadAtStageMin**.

---

Read-only property  
Primary script: Yes  
Secondary: No

### ExtraExtnAxis *n,v*

This function is used to add a new axis to the graph during the test. The axis will be a transformation of the Extension axis. For example:

ExtraExtnAxis "Tear Propagation", 0.5

See also ExtraLoadAxis.

---

Read-only property  
Primary script: Yes  
Secondary: No

### ExtraLoadAxis *n,v*

This function is used to add a new axis to the graph during the test. The axis will be a transformation of the Load axis. For example:

ExtraLoadAxis "Strength", [ [1] / Width ]

## 6.0 REFERENCE

This line will add an axis called Strength which will show the Load over the sample Width.

See also ExtraExtnAxis.

---

Read-only property  
Primary script: Yes  
Secondary: No

### GetExtraResult *n*

This function is used to lookup the value of an Extra Result. You can use these values in you calculations. For example:

Set M = GetExtraResult ("Mass")

---

Function  
Primary script: Yes  
Secondary: Different

### Global *n*, *v*

This function is used in the primary script to create a test-global variable. The parameter *n* must be string which is the name of the variable, and the parameter *v* is its value.

The **Global** function behaves differently in the secondary script, where it retrieves test-global variables.

Normal script variables will be lost at the end of each script, therefore you need to use the test-global variable to store values between the primary and secondary scripts.

Test-global variables are required to perform calculations which involves one value that can only be measured in the primary script, and a second value which is only available in the secondary. For example, the only way of measuring the load reached by a specific stage in the middle of a multi-stage test is to use the LoadAtStageMax property at the end of that stage, in the primary script. This can be achieved as follows:

**Global "L1", LoadAtStageMax**

This line stores the maximum load over the previous stage into the test-global variable "L1". The example is continued in the next section.

---

Function  
Primary: Different  
Secondary: Yes

### Global(*n*)

## 6.0 REFERENCE

This function is used in the secondary script to retrieve test-global variables created in the primary script. The parameter  $n$  must be a string which is a name which matches one created in the primary script.

Continuing the example of the previous section, the following script calculates the difference in load between a marker load and the test-global variable "L1".

```
Result "F", [MarkerLoad(1)-Global("L1")]
```

In this case it is necessary to use test-global variables because the stage load can only be measured in the primary script, and the marker load can only be measured in the secondary script.

---

Read-only property  
Primary script: Yes  
Secondary: Yes

### Height

This property stores the height of the sample, provided the sample height option on the first page of the Ondio setup is set to anything other than "Not Applicable".

This property is useful for crushing a sample by a fraction of its height, for example:

```
Stage [Height*[0.5]], [1mm/s]
```

This property is also useful for reporting results in strain units, for example:

```
Result "Max Strain", [ExtnAtMax/Height]
```

---

Read-only property  
Primary script: Yes  
Secondary: Yes

### InputAtMax c

This function requires a Data Acquisition Card, and returns the value of the analogue input on channel  $c$ , at the point of maximum load. If used part way through the test, it returns the input at the point of maximum load so far.

For example:

```
Result "Max Volts", InputAtMax("ch1")
```

## 6.0 REFERENCE

See also **InputAtMin**, **InputAtStageMax**,  
**InputAtStageMin**.

---

Read-only property  
Primary script: Yes  
Secondary: Yes

### **InputAtMin c**

This function requires a Data Acquisition Card, and returns the value of the analogue input on channel *c*, at the point of minimum load. If used part way through the test, it returns the input at the point of minimum load so far.

For example:

```
Result "Min Volts", InputAtMin("ch1")
```

See also **InputAtMax**, **InputAtStageMax**,  
**InputAtStageMin**.

---

Read-only property  
Primary script: Yes  
Secondary: Yes

### **InputAtStageMax c**

This function requires a Data Acquisition Card, and returns the analogue input value on channel *c* at the point of maximum load over the previous stage.

For example:

```
Stage [10mm], [60mm/s]
```

```
Result "Max V", InputAtStageMax("ch1")
```

See also **InputAtStageMin**.

---

Read-only property  
Primary script: Yes  
Secondary: Yes

### **InputAtStageMin c**

This function requires a Data Acquisition Card, and returns the analogue input value on channel *c* at the point of minimum load over the previous stage.

For example:

```
Stage [10mm], [60mm/s]
```

```
Result "Min V", InputAtStageMin("ch1")
```

## 6.0 REFERENCE

See also **InputAtStageMax**.

---

Read-only property  
Primary script: Yes  
Secondary: No

### Load

This property stores the current load.

See also **Extn**, **Work**, and **TestTime**.

More precisely, it stores the load at the end of the previous stage, therefore it does not change until you run another stage.

---

Read-only property  
Primary script: No  
Secondary: Yes

### LoadAtBreak

This property stores the load at the break point. It is only available if the sample did break, therefore you must check the **Broken** property first.

See also **ExtnAtBreak**, **WorkToBreak**.

This property may not hold accurate results if you have used the **BreakOn** or **BreakOff** commands in the primary script.

---

Read-only property  
Primary script: No  
Secondary: Yes

### LoadAtGreatestSlope

This property contains the Load at the Modulus.

See also **ExtnAtGreatestSlope**.

---

Read-only property  
Primary script: Yes  
Secondary: Yes

### LoadAtMax

This property stores the maximum load over the whole test. If the property is used before the test has finished, it contains the maximum load so far.

See also **ExtnAtMax**, **LoadAtStageMax**

## 6.0 REFERENCE

---

Read-only property  
Primary script: Yes  
Secondary: Yes

### LoadAtMin

This property stores the minimum load over the whole test. If the property is used before the test has finished, it contains the minimum load so far.

See also **ExtnAtMin**, **LoadAtStageMin**

---

Read-only property  
Primary script: No  
Secondary: Yes

### LoadAtOffsetYield

This property stores the load at the offset yield point. For details about the offset yield point, see the **SetOffsetExtn** function.

See also **ExtnAtOffsetYield**.

---

Read-only property  
Primary script: Yes  
Secondary: No

### LoadAtStageMax

This property stores the maximum load over the previous stage.

See also **ExtnAtStageMax**.

---

Read-only property  
Primary script: Yes  
Secondary: No

### LoadAtStageMin

This property stores the minimum load over the previous stage.

See also **ExtnAtStageMin**.

---

Function  
Primary script: No  
Secondary: Yes

### MarkerExtn(*n*)

The parameter *n* must be the number of a marker. This function will return the extension value at the point where the user placed that marker.

For example to report the difference in extension between two marker points, use a script similar to the following:

---

## 6.0 REFERENCE

**Result** "E", [**MarkerExtn**(2)-**MarkerExtn**(1)]

**NEXYGEN** will rerun the secondary script if any of the markers are moved.

See also **MarkerLoad** and **MarkerWork**.

---

Function  
Primary script: No  
Secondary: Yes

### **MarkerInput**(*n*,*c*)

The parameter *n* must be the name of a marker added using the **MarkerNow** function. Parameter *c* is the name of the analogue input channel. This function will return the analogue input value at the point where the user placed that marker.

For example to report the difference in value between two marker points on channel "ch1", use a script similar to the following:

**Result** "R", [**MarkerInput**("m1", "ch1") -  
**MarkerInput**("m2", "ch1")]

**NEXYGEN** will rerun the secondary script if any of the markers are moved.

See also **MarkerLoad**, **MarkerWork**, **MarkerTime** and **MarkerExtn**.

---

Function  
Primary script: No  
Secondary: Yes

### **MarkerLoad**(*n*)

The parameter *n* must be the number of a marker. This function will return the load value at the point where the user placed that marker.

For example to report the difference in load between two marker points, use a script similar to the following:

**Result** "L", [**MarkerLoad**(2)-**MarkerLoad**(1)]

**NEXYGEN** will rerun the secondary script if any of the markers are moved.

See also **MarkerExtn** and **MarkerWork**.

## 6.0 REFERENCE

---

Function  
Primary script: Yes  
Secondary: Yes

### MarkerNow(*n*)

In the primary script, this function is used to add a Marker to the graph during the test. In the secondary script, the marker is placed at the end of the test.

The markers have user-defined names. For example:

```
Stage [10mm], [2mm/s]
```

```
MarkerNow ("Stage1End")
```

```
Stage [0mm], [2mm/s]
```

```
MarkerNow ("Stage2End")
```

### Handling of duplicate marker names

In the primary script, a duplicate marker name in MarkerNow has the effect of moving the old marker to the new location. In the secondary script, the request to place the new marker is ignored.

See also **MarkerLoad**, **MarkerExtn**, **MarkerWork** and **MarkerInput**.

---

Function  
Primary script: No  
Secondary: Yes

### MarkerTime(*n*)

The parameter *n* must be the name of a marker added using the **MarkerNow** function. This function will return the time value at the point where the user placed that marker.

For example to report the difference in time between two marker points, use a script similar to the following:

```
Result "T", [MarkerTime("m1") -  
MarkerTime("m2")]
```

**NEXYGEN** will rerun the secondary script if any of the markers are moved.

See also **MarkerLoad**, **MarkerWork**, **MarkerExtn** and **MarkerInput**.

## 6.0 REFERENCE

---

Function  
Primary script: No  
Secondary: Yes

### MarkerWork(*n*)

The parameter *n* must be the number of a marker. This function will return the amount of work done on the sample from the start of the test up to the point where the user placed that marker.

For example to report the amount of work done between two markers, use a script similar to the following:

```
Result "W", [MarkerWork(2)-MarkerWork(1)]
```

**NEXYGEN** will rerun the secondary script if any of the markers are moved.

See also **MarkerLoad** and **MarkerExtn**.

---

Read-only property  
Primary script: No  
Secondary: Yes

### Modulus

When this property is accessed it will automatically locate the steepest slope on the trace, marking the graph with a diagonal line.

The value of this property is the gradient of this line, the Tangential Modulus.

This value is in Load/Extension units. If you want to report the Modulus of Elasticity (Modulus in Stress/Strain units) then you will need to use a calculation similar to:

```
Result "MOE", [Modulus*Height/Area]
```

---

Function  
Primary script: Yes  
Secondary: No

### Pause *t*

This function is used to pause the test. For example:

```
Pause [30 min]
```

---

Read-only property  
Primary script: Yes  
Secondary: Yes

### Progress *m*

## 6.0 REFERENCE

This function can be used to add a progress message to the status bar of the graph. The same message will be displayed on the **NEXYGEN** screen saver.

The message *m* must be a string.

This feature is particularly useful for tests of very long duration. For example, the following script displays the cycle number on the status bar:

```
for i=1 to 100000
  Progress "Cycle number " & i
  Stage [20mm],[1mm/min]
  Stage [10mm],[1mm/min]
next
```

---

Function  
Primary script: Yes  
Secondary: Yes

### Result *n*, *v*

This function creates a new result named by the string *n*. The result is given the value *v*.

---

Function  
Primary script: Yes  
Secondary: No

### RunAllStages

This function is equivalent to repeatedly calling the **RunStage** function until there are no more stages left.

Like **RunStage**, this function is useful if you want to run successive stages without pauses. It is more convenient than **RunStage** if you have lots of stages, and you do not need to take intermediate measurements.

For example, the following script will create over 60 stages to move the machine in a sine wave motion. This may be useful preconditioning for some types of sample.

```
for i =0 to 2*3.14 step 0.1
  set l=[[100mm]*sin(i)]
  set s=[[10mm/s]*abs(cos(i))]
  SetupStage l,s
next
RunAllStages
```

## 6.0 REFERENCE

Function  
Primary script: Yes  
Secondary: No

### RunStage

This function is used in combination with the functions **SetupStage** and **SetupStageHold**. When used together they provide the means to run successive stages without any pauses.

Use the **SetupStage** or **SetupStageHold** functions to send the stage information down to the machine, without starting its movement. When sufficient stages have been sent, use the **RunStage** command to run the first of them.

The **RunStage** function will not finish until the machine reaches the limit set for the first stage. Upon reaching the limit, all of the stage properties such as **Load**, **Extn**, **LoadAtStageMax** etc. are updated, and the machine will start to move along the second stage promptly.

The machine starts to move along the second stage in order to prevent any undesirable pauses. This happens at the same time as the next line in your primary script starts to run. All of the stage properties contain values measured at the stage limit, so you can store them in variables for later calculation.

Although the machine has started to move along the second stage, it will not continue for long unless you call the **RunStage** function for a second time to confirm this stage. Again, this function will not finish until the second limit is reached.

For example, this function must be used to write a test which will cycle quickly, and measure the sample load at the peak of each cycle.

## 6.0 REFERENCE

```
SetupStage [10mm],[1mm/s]
SetupStage [0mm] ,[1mm/s]
SetupStage [10mm],[1mm/s]
SetupStage [0mm] ,[1mm/s]
RunStage
set 11=Load
RunStage
RunStage
set 12=Load
RunStage
```

This test starts by sending four stages down to the machine. The first **RunStage** function call sends the machine moving to the limit of the first stage, 10mm, and turns it around to head down to the limit of the second, 0mm. The variable **11** is set to the load at the 10mm point.

The following two **RunStage** function calls send the machine to the limits of the second and third stages, 0mm and then 10mm, and turns it around towards the destination of the fourth.

Finally **12** is set to the load at the third stage limit, then the script waits for the machine to actually reach the fourth limit before finishing.

---

Function  
Primary script: No  
Secondary: Yes

### **SetDigitalHigh *p,c***

This function requires a Data Acquisition Card, and sets the digital output high on port *p*, line *c*. For example, we want to turn on an external switch at the end of a stage:

```
Stage [10mm], [2mm/s]
SetDigitalLow 0,0
```

Make sure the selected port supports digital outputs.

See also **SetDigitalLow**.

---

Function  
Primary script: No  
Secondary: Yes

### **SetDigitalLow *p,c***

## 6.0 REFERENCE

This function requires a Data Acquisition Card, and sets the digital output low on port p, line c. For example, we want to turn off an external switch at the end of a stage:

```
Stage [10mm], [2mm/s]
```

```
SetDigitalLow 0,0
```

Make sure the selected port supports digital outputs.

See also **SetDigitalHigh**.

---

Function  
Primary script: No  
Secondary: Yes

### **SetOffsetExtn e**

The parameter *e* must be an extension, which is the offset value used in the automatic offset yield calculation, also called the proof strength point.

In most cases you want the offset to be specified as a fraction of the sample gauge length. In this case, you will use the following script to locate the 1% proof stress point.

```
SetOffsetExtn [Height*0.01]
```

After this, the load and extension values at this point are available in the properties **ExtnAtOffsetYield** and **LoadAtOffsetYield**. Ondio will automatically mark the point on the graph with an event and a diagonal line.

---

Function  
Primary script: Yes  
Secondary: No

### **SetupStage l, s**

The usual means of moving the machine is with the **Stage** command.

**SetupStage** is a variation of the **Stage** command which gives you greater flexibility. This is particularly useful for eliminating the small pauses between stages that are unavoidable when using **Stage**.

This function prepares the machine for movement by sending the stage information down to the machine, but it does not start it moving. You can call this function many times, once for each stage that you want to run. The

## 6.0 REFERENCE

machine is able to run them all without pauses because it has advance notice of what it needs to do.

Once your stages have been sent to the machine using this **SetupStage** function you can start the machine moving by calling the **RunStage** or **RunAllStages** function.

---

Function  
Primary script: Yes  
Secondary: No

### SetupStageHold *l, s, t*

This function is similar to **SetupStage**, but on reaching the limit *l* the machine will hold for a time *t*.

The behavior of this hold is described under the StageHold function.

---

Function  
Primary script: Yes  
Secondary: No

### Stage *l, s*

This function causes the machine to move to the limit condition specified by the parameter *l*. It moves at a speed *s*.

This function will not finish until the machine reaches that limit. Upon reaching the limit, all of the stage properties such as **Load**, **Extn**, **LoadAtStageMax** etc. are updated.

Note that if the sample has already broken before this stage is started, or if the End or Advance buttons have been pressed, then the machine will not move and the properties not updated.

Calling this function is equivalent to calling the two functions:

**SetupStage *l, s***  
**RunAllStages**

The limit *l* can be either a load limit or an extension limit.

The speed *s* can be an extension rate or a load rate, if supported by the machine.

## 6.0 REFERENCE

---

Function  
Primary script: Yes  
Secondary: No

### StageHold *l, s, t*

This function performs a similar function to **Stage**, but on reaching the limit *l* the machine will hold for a time *t*. If the limit *l* is an extension then the machine will hold that extension. If the limit *l* is a load then it will hold that load, if supported by the machine.

Calling this function is equivalent to calling the two functions:

```
SetupStageHold l,s,t  
RunAllStages
```

---

Read-only property  
Primary script: Yes  
Secondary: No

### StageNumber

This property stores the number of stages run so far. Before any stages are run, it will contain zero.

---

Function  
Primary script: Yes  
Secondary: No

### StageStaticExtn(*l*)

This function examines the previous stage to locate where the trace passed through the load value *l*. It returns the extension value at that point.

Unlike the similar function **StaticExtn**, it does not look at the whole graph, just the part of the graph from the previous stage.

---

Function  
Primary script: Yes  
Secondary: No

### StageStaticLoad(*e*)

This function examines the previous stage to locate where the trace passed through the extension value *e*. It returns the load value at that point.

Unlike the similar function **StaticLoad**, it does not look at the whole graph, just the part of the graph from the previous stage.

## 6.0 REFERENCE

Function  
Primary script: Yes  
Secondary: Yes

### StaticExtn(*l*)

This function examines the graph to locate where the trace passed through the load value *l*. It returns the extension value at that point.

If the graph passes through that load value more than once it will return the first intercept.

See also **StaticLoad** and **StageStaticExtn**

For example, this function can be used to calculate a chord modulus, that is the gradient of a line that intercepts the graph at two load values.

```
set e1=StaticExtn([60N])
set e2=StaticExtn([80N])
Result "Modulus", [(e2-e1)/[20N]]
```

Function  
Primary script: Yes  
Secondary: Yes

### StaticLoad(*e*)

This function examines the graph to locate where the trace passed through the extension value *e*. It returns the load value at that point.

If the graph passes through that extension value more than once, such as in a cycle test, it will return the first intercept.

See also **StaticExtn** and **StageStaticLoad**

For example, this function can be used to calculate the average load during a tearing test. The following script calculates the average of loads measured at five points during the tear.

```
set a=StaticLoad([1mm])
set b=StaticLoad([3mm])
set c=StaticLoad([5mm])
set d=StaticLoad([7mm])
set e=StaticLoad([9mm])
Result "Average", [(a+b+c+d+e)/[5]]
```

## 6.0 REFERENCE

---

Function  
Primary script: Yes  
Secondary: Yes

### StaticWork(e)

This function returns the static work at a point on the graph.

The parameter is either the load or the extension at the point at which the Work is required. The static work is calculated as the area under the Load-Extension graph between the start of the test and the requested load or extension. The value of the load or extension corresponding to the parameter passed, is determined through the common intercept on the time axis.

For example:

```
Set w1 = StaticWork([10mm])
```

```
Set w2 = StaticWork([2N])
```

See also **MarkerWork**

---

Function  
Primary script: Yes  
Secondary: No

### StopMachine

You will only need to use this function if you are writing an advanced test where your primary script will spend a long time where it does not need to control the machine. For example, a test which involves exercising the sample, turning on a heater, waiting for the temperature to stabilize, then exercising the sample further.

Call this function if the machine has been moving, and you will not be moving it again for more than approximately 10 seconds. Note that it is not necessary to call this function at the end of the test.

---

Function  
Primary script: Yes  
Secondary: No

### TearOn

**NEXYGEN**'s normal operating mode includes a self-tuning algorithm for deciding how many data points to store from the load and extension signals. This algorithm ensures that sufficient data is stored for the accuracy required, while

## 6.0 REFERENCE

optimising performance of your computer and materials testing system as a whole.

This function switches **NEXYGEN** into a different mode which is required for tearing tests, where fine detail in the load channel is significant for the whole test.

You should call this function at the beginning of a tearing test.

---

Function

Primary script: Yes

Secondary: No

### TearOff

This function returns **NEXYGEN** to its normal operating mode, from the tearing-test mode started using the **TearOn** function.

For example, this function would be required for a test involving tearing a fabric, then holding load to allow it to creep. Use the **TearOn** function at the start of the test to capture all the detail during tearing, then use **TearOff** before holding load to avoid wasting memory and hard disk space.

**TearOn**

**Stage** [20mm], [1mm/s]

**TearOff**

**set** l=[LoadAtStageMax\*[0.5]]

**StageHold** l, [1mm/s], [120min]

---

Read-only property

Primary script: Yes

Secondary: No

### TestTime

This property stores the time since the test began. This is a value with units, and is not compatible with time values from functions such as VBScript's "Time", where you always have to remember that the number is in seconds.

Note that this property stores test time, not computer time. The property does not change until you run another stage.

See also **Load**, **Extn**, and **Work**.

---

Function  
Primary script: Yes  
Secondary: No

### UsePercentageDropDetector

This function is used to set the Break Detector to look for a percentage drop in the load.

Calling this function will also switch the break detector on as with the BreakOn function.

See also UseSharpDropDetector .

---

Function  
Primary script: Yes  
Secondary: No

### UseSharpDropDetector

This function is used to set the Break Detector to look for sharp drops in the load.

Calling this function will also switch the break detector on as with the BreakOn function.

See also UsePercentageDropDetector.

---

Function  
Primary script: Yes  
Secondary: No

### ValueOfInput c

This function requires a Data Acquisition Card, and returns the current value of analogue input channel c. For example:

```
Stage [10mm], [2mm/s]
```

```
Result "r1", ValueOfInput("ch1")
```

---

Read-only property  
Primary script: Yes  
Secondary: Yes

### Width

This property stores the width of the sample, if that option is chosen in the first page of Ondio's setup.

In most cases, the **Area** property is more convenient.

---

Read-only property  
Primary script: Yes  
Secondary: Yes

### Work

## 6.0 REFERENCE

This property stores the work done on the sample so far. See also **Load**, **Extn**, and **TestTime**.

More precisely, it stores the work done up to the end of the previous stage, therefore it does not change until you run another stage.

---

Read-only property  
Primary script: No  
Secondary: Yes

### **WorkToBreak**

This property stores the work done on the sample up to the break point. It is only available if the sample did break, therefore you must check the **Broken** property first.

See also **LoadAtBreak**, **ExtnAtBreak**.

This property may not hold accurate results if you have used the **BreakOn** or **BreakOff** commands in the primary script.

## 7.0 ADVANCED SCRIPTS

The Tutorial and Reference sections of this manual contain all of the information necessary to make full use of **NEXYGEN** Ondio.

This section is intended for advanced users who have some familiarity with other scripting and programming languages, who may benefit from an explanation of some Ondio features in a general programming context.

Because these are advanced topics, Lloyd Instruments may not be able to provide technical support for all of the issues discussed in this section.

---

### 7.1 THE VBSCRIPT LANGUAGE

The script language used by default for Ondio scripts is the VBScript programming language produced by Microsoft. This language is used in many products from Microsoft and other Vendors, and is becoming the standard script language in Windows.

If you are familiar with one of the other languages in the VB family, such as Visual Basic or VBA, or indeed any other form of BASIC programming language, then you will find the full VBScript language very easy to learn.

Some documentation for VBScript can be installed from the Ondio CD, from the directory:

Microsoft\VBScript\vbsdoc.exe

The standard VBScript language has been extended to include a value-with-units data type, and unit-tracking arithmetic. This section explains the units-tracking enhancements in detail.

The tutorial section of this manual only covers a small subset of the whole VBScript programming language, a subset which is sufficient for the majority of materials testing needs. However all of the features of VBScript are available if you need them.

Finally, this section explains how Ondio can be used with a different script language. While VBScript is a practical language for writing small scripts, it may not be the best

## 7.0 ADVANCED SCRIPTS

solution if you have need of a complicated script and are already familiar with a different language.

---

### 7.2

### UNIT-TRACKING EXTENSIONS

Ordinary VBScript has only one data type, the Variant. It has several sub-types for storing different types of value, ranging from simple values such as integers and strings up to higher-level values such as dates, and currency values. Unfortunately, a Variant does not normally have a sub-type for storing a value with units.

Ondio extends VBScript capabilities in three ways:

- 1) It provides a way of storing values with units in a variant.
- 2) It provides a way to write a value with units directly in a script, for example [10mm].
- 3) It provides unit-tracking arithmetic.

The first of these is handled in a straightforward manner. Ondio stores values with units as the object sub-type of a Variant. The built-in properties which are values with units are quite simply object properties.

In VBScript it is an error to use an arithmetic operator ( + - \* or / ) or comparison operator ( < or > ) on an object. Therefore all unit-tracking arithmetic must be surrounded by square brackets.

Before running a script, Ondio scans the text for square brackets, replacing arithmetic and comparison operators with calls to several built-in functions which are normally hidden from view.

Note that square brackets are not used for any other purpose in the VBScript language.

Table 1 lists the operators that are changed if they appear between square brackets, and the functions that they are replaced with:

## 7.0 ADVANCED SCRIPTS

$x + y$	<code>qadd(x,y)</code>
$x - y$	<code>qsub(x,y)</code>
$x * y$	<code>qmul(x,y)</code>
$x / y$	<code>qdiv(x,y)</code>
$x < y$	<code>qlt(x,y)</code>
$x > y$	<code>qgt(x,y)</code>

Table 1

Therefore, the following Ondio script fragments in Table 2 are equivalent, and indeed the fragments on the left would be replaced by the fragment on the right before being run by VBScript.

<code>[a+b]</code>	<code>qadd(a,b)</code>
<code>[a+c*d]</code>	<code>qadd(a,qmul(c,d))</code>
<code>[(a+d)*e]</code>	<code>qmul(qadd(a,d),e)</code>
<code>[a*b/c]</code>	<code>qdiv(qmul(a,b),c)</code>

Table 2

If the square brackets contain a value with units, rather than an expression, then the square brackets are converted into a call to the built-in function `qscale`, and the contents of the square brackets converted to a string. The following fragments are equivalent:

<code>[10mm]</code>	<code>qscale("10mm")</code>
<code>[a+[10mm]]</code>	<code>qadd(a,qscale("10mm"))</code>

Table 3

These built-in functions are always available, and a script can use them at any time like any other built-in function. However, the square-bracket notation is preferred because it is clearer and more concise.

---

### 7.3

### OTHER SCRIPT LANGUAGES

If you are already familiar with a different script language and you are intending to write a large or complicated test in Ondio, then you may prefer to use your favorite language rather than the default, VBScript. Before taking this step there are some disadvantages that you should be aware of:

- 1) Lloyd Instruments can only provide technical support when using the VBScript language. If you are using a

## 7.0 ADVANCED SCRIPTS

different language and encounter a problem, you will be asked to try to reproduce the problem using VBScript before Lloyd Instrument's technical support can help.

- 2) Lloyd Instruments can only provide programming assistance for VBScript.
- 3) If your language already uses square brackets then you will not be able to use the Ondio unit-tracking extensions. You will need to use the expanded functions (**qadd**, etc) directly.

### CHOOSING A LANGUAGE

Ondio can make use of any ActiveX Scripting Language. The most familiar use of ActiveX Scripting is for scripting of Internet Web pages. In general, if a language supports scripting of Microsoft Internet Explorer using HTML similar to `<SCRIPT LANGUAGE = "VBScript">` then it will be possible to use it with Ondio too.

The language is chosen using the Script Language command on the Edit menu. You will see the default is "VBScript". You can choose any language from the list of registered languages, which should include at least JScript as well as VBScript. If your favorite language is not on the list you will need to type the name of the language into that box, the same name as is used in the `<SCRIPT LANGUAGE = "VBScript">` HTML.

On this box you can also choose whether or not the square-bracket preprocessing should occur. Most languages other than VBScript do use square brackets, and therefore it is likely that you will need to turn off this option and use the expanded functions (**qadd**, etc) directly.

### GLOBAL FUNCTIONS

Ondio provides approximately 70 built-in properties and functions. All of these are *global*, meaning that you can use them anywhere in your script.

Some ActiveX Script languages do not permit that many functions to be made global. In these cases Ondio's

## 7.0 ADVANCED SCRIPTS

functions and properties are available inside one global library, under the name "Ondio".

For example, if you are using the language Python the built-in property **LoadAtMax** can be accessed as **Ondio.LoadAtMax**

---

### 7.4

### STOPPING THE MACHINE

Ondio will automatically stop the machine after running the **Stage** function, after **RunAllStages**, or after **RunStage** when there are not more stages left.

However, you will need to take a further manual step if your advanced test will be performing some other activity in the primary script which will take longer than approximately 10 seconds before moving the machine again. Examples of such an activity might be:

- 1) The script waiting for the temperature of a thermal cabinet to stabilize.
- 2) Performing a long calculation in the script.
- 3) The script fetching data from an external database.
- 4) The script performing a long calculation in a different program, such as a spreadsheet.

Before performing this activity you should call the **StopMachine** function, described in detail on page 89.

If you do not call this function the machine will not be completely stationary, since it is waiting in anticipation of the next stage command. Calling this function will allow the machine to power down its drive and apply brakes if present. For some machines, it may also prevent glitches during the period when it is supposed to be stationary.

## **8.0 END USER LICENCE AGREEMENT**

This Software Licence Agreement ("Agreement") is made and entered in to by and between Lloyd Instruments Limited (herein referred to as Lloyd Instruments) and the Customer, hereinafter referred to as the "Licencee". Lloyd Instruments hereby grants and the Licencee hereby accepts a non-exclusive, non-transferable Licence to use the Licenced Material, subject to the following terms and conditions.

---

### **8.1 DEFINITIONS**

"Licenced Material" is each product furnished by Lloyd Instruments to the Licencee and includes supporting materials and any related updated product or product portion furnished by Lloyd Instruments.

"User System" is a single CPU.

---

### **8.2 LICENCE**

The Licence granted hereunder authorises the Licencee, on a non-exclusive basis, to use the Licenced Material for use on computers which are interfaced with instruments manufactured or sold by Lloyd Instruments. The use by the Licencee of Licenced Material supplied by Lloyd Instruments shall be deemed to be an acceptance of the terms of this Licence Agreement by both parties.

---

### **8.3 REPRODUCTION RIGHTS**

Licencee may make one copy of the Licenced Materials for archival (backup) purposes. Licencee may copy the Licenced Material on to a hard disk which is the integral hardware component of the single computer on which the Licenced Material is to be used. In this case, the original delivery media becomes the backup copy and the working copy is that which resides on the hard disk.

## **8.0 END USER LICENCE AGREEMENT**

Licencee agrees not to remove Lloyd Instruments' copyright notices and other legends both in and on every copy of the Licenced Material, in any form, including partial copies.

Licencee must not modify the Licenced Material or incorporate it into any other software without the express written permission of Lloyd Instruments.

---

### **8.4 SECURITY AND TRANSFERABILITY**

The Licencee agrees not to provide, sub licence, use in a time-sharing service or otherwise make available the Licenced Material or any portion thereof, in any form, to any person other than the Licencee or Lloyd Instruments' employees without the prior written consent of Lloyd Instruments.

The Licencee agrees to take all reasonable steps, both during and after the terms of this Licence Agreement, to ensure that no unauthorised person shall have access to the Licenced Material, and that no unauthorised copy, in whole or in part, shall be made.

---

### **8.5 TITLE**

Nothing contained in this Licence Agreement shall directly or indirectly, by implication or otherwise, be construed as an assignment or grant to the Licencee of any right, title or interest in the copyrights, patents or trade secrets of Lloyd Instruments', all rights related thereto being reserved by Lloyd Instruments except for the non-exclusive Licence granted hereunder to the Licencee as expressly provided herein.

---

### **8.6 WARRANTY**

Lloyd Instruments warrants for a period of ninety (90) days from the date of delivery to the Licencee that the Licenced Materials will conform to Lloyd Instruments'

## **8.0 END USER LICENCE AGREEMENT**

published specifications and release current at the time of delivery. Lloyd Instruments' sole obligation, and the Licencee's sole remedy, shall be for Lloyd Instruments to exert its reasonable efforts to correct such alleged defects and to supply the Licencee with a changed version within a reasonable time after Licencee notifies Lloyd Instruments in writing of any alleged defect. This warranty does not cover modifications to the Licenced Material made by any person not employed by Lloyd Instruments, or any defects caused by or otherwise related to such modifications. Except for the express warranty stated above, Lloyd Instruments grant no warranties, either express or implied, on any licenced material including, but not limited to, all warranties of merchantability and fitness for a particular purpose. Licencee shall bear the risk of loss or damage to the Licenced Material whilst under its control.

---

### **8.7 LIABILITY**

Lloyd Instruments will not be responsible for any direct, incidental or consequential damages, including, but not limited to Loss of Profits or in any way for any liability arising out of this Agreement, resulting from the use of the Licenced Materials save that nothing in this Agreement affects Lloyd Instruments' liability for death or personal injury arising out of Lloyd Instruments' negligence.

---

### **8.8 TERMS AND TERMINATION**

The terms of this Agreement and any licence granted hereunder shall begin on the date of supply by Lloyd Instruments of the Licenced Material, and shall run until the Licencee discontinues use of all Licenced Materials, so long and only during such period, as the Licencee complies with the terms and conditions set forth herein. This Agreement and any licence granted hereunder may be terminated by Lloyd Instruments, in addition to any other remedies that it may have, if Licencee fails to comply with any of the terms and conditions of this Agreement, provided written notice describing the default is given to the Licencee. The

## **8.0 END USER LICENCE AGREEMENT**

Licencee shall have a period of thirty (30) days from receipt of such notice to certify in writing that it has remedied the default. If Lloyd Instruments does not receive such certification within this thirty (30) day period, Lloyd Instruments may give the Licencee notification of termination. Within ten (10) days of notification of such termination or within thirty (30) days after the Licencee has discontinued use of all Licenced Material, Licencee shall destroy or return to Lloyd Instruments all the Licence Material and all reproductions in whole or in part in Licencee's possession or under Licencee's control and shall so certify in writing to Lloyd Instruments.

If the Licencee has been permitted to modify the Licenced Material then the Licencee shall remove the modifications from the Licenced Material before so destroying or returning the Licenced Material.

Lloyd Instruments shall have the right to terminate this Agreement immediately if the Licencee goes into liquidation or has a receiver or manager appointed over any of its assets or undertakings or is unable to pay its debts as defined in Section 123 of the Insolvency Act 1986 or, being an individual, commits an act of bankruptcy or dies.

---

### **8.9**

#### **GENERAL**

Any variation of this Agreement shall be in writing and signed by or on behalf of the parties.

Each party acknowledges that in entering into this Agreement it does not rely on any representation warranty or other provision except as expressly provided herein.

No delay or omission of either party in exercising any right hereunder shall impair such right or is to be construed as a waiver thereof.

This Agreement shall be construed and take effect according to English Law and be subject to true jurisdiction of the English Courts.