MetroScript™ Programming Language

OMP-0399Q

Zygo Corporation
Laurel Brook Road
P.O. Box 448
Middlefield, Connecticut 06455

Telephone: (860) 347-8506
E-mail: inquire@zygo.com
Website: www.zygo.com

ZYGO CUSTOMER SUPPORT

For help within North America, please use the contacts listed below. For help in other countries, please contact your local Zygo Corporation representative. Be sure to supply the instrument model and serial numbers, and the software version.

PHONE Monday - Friday, 8 a.m. - 8 p.m. (Eastern Standard Time)

(800) ZYGO NOW (800) 994-6669

or (860) 704-5191

FAX (860) 346-4188

INTERNET www.zygo.com
support@zygo.com

WRITE Zygo Corporation
Laurel Brook Road
P.O. Box 448
Middlefield, CT 06455-0448
Attn: Customer Support

MANUAL REVISION INFORMATION

The document (OMP) number and the applicable revision letter for this manual appear on the title page. The publication date appears below.

Revision	Release Date	Revision	Release Date
A (software v. 7.2.0)	March 1998	J (software v. 7.14.0)	September 2003
B (software v. 7.3.2)	January 2000	K (software v. 7.14.0)	January 2004
C (software v. 7.4.2)	Nov. 2000	L (software v. 7.15.0)	March 2004
D (software v. 7.4.2)	April 2001	M (software v. 8.0.0)	July 2004
E (software v. 7.6.0)	August 2001	N (software v. 8.0.2, 8.1.0, 8.11)	June 2005
F (software v. 7.7.0)	March 2002	P (software v. 8.11)	August 2005
G (software v. 7.10.0)	Nov. 2002	Q (software v. 8.2.0, 8.3.0, 8.3.2, 8.3.3, 9.0)	March 2011
H (software v. 7.12.0)	July 2003		

MANUAL NOTATIONS

Warning!

Denotes a hazard that could cause injury to personnel, and/or damage to the equipment.

Note, provides helpful information.

ZYGO Software License Agreement

The following is a legal agreement between you and ZYGO Corporation. This software is licensed to you and not sold. You may use this software only according to the terms of this License.

1. **GRANT OF LICENSE.** You may use the software only on a single computer at a time. You may not network the software or otherwise use it on more than one computer or computer terminal at the same time. The License covers all users on the single system.

2. **OWNERSHIP OF SOFTWARE.** The software is owned by ZYGO and is protected by United States copyright laws and international treaty provisions. Therefore, you must treat the software like any other copyrighted material.

3. **USE RESTRICTIONS.** You may transfer the software to a hard disk and make copies of the software solely for backup or archival purposes. You may not alter, modify, or adapt any part of the software or documentation. This means you may not reverse-engineer, decompile, disassemble this software, or create derivative works from it.

4. **TRANSFER RESTRICTIONS.** This Software is licensed only to you, and may not be transferred to anyone without prior written consent of ZYGO. The terms and conditions of this Agreement shall bind any authorized transferee of software. In no event may you transfer, assign, rent, lease, sell, or otherwise dispose of the software on a temporary or permanent basis except as expressly provided herein.

5. **TERMINATION.** This License is effective until terminated. This License will terminate automatically without notice from ZYGO if you fail to comply with any provision of this License. Upon termination you shall destroy the written materials and all copies of the software, including modified copies, if any.

Disclaimer of Warranty on Software

THE SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, AND ZYGO EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES. ZYGO DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE OF OR THE RESULTS OF THE USE OF THE SOFTWARE OR ANY ACCOMPANYING WRITTEN MATERIALS IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS, OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE AND WRITTEN MATERIALS IS ASSUMED BY YOU.

Software Limited Warranty

ZYGO warrants the magnetic media on which the software is recorded to be free from defects in materials and faulty workmanship under normal use for a period of 90 days from the date of delivery. ZYGO will replace the media, provided you return the faulty media to ZYGO with return authorization. ZYGO shall have no responsibility to replace magnetic media damaged by accident, abuse, or misapplication.

Limitation of Liability

IN NO EVENT SHALL ZYGO, OR ITS EMPLOYEES AND AFFILIATES BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, INCIDENTAL, OR DIRECT DAMAGES ARISING OUT OF THE USE OR THE INABILITY TO USE THE SOFTWARE OR ACCOMPANYING WRITTEN MATERIALS, EVEN IF ZYGO OR A ZYGO REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHERS, WHICH VARY FROM STATE TO STATE.

U.S. Government Restricted Rights

The software and documentation are provided with restricted rights. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at 52.227-7013. Contractor/manufacturer is Zygo Corporation, Laurel Brook Road, Middlefield, CT 06455.

Should you have any questions concerning this Agreement, or if you wish to contact ZYGO for any reason, please write: ZYGO Service Department, Laurel Brook Road, P.O. Box 448, Middlefield, CT 06455-0448.

11/2002

CHAPTER 2 – METROSCRIPT FUNDAMENTALS-CONTINUED

CHAPTER 3 – EXAMPLE SCRIPTS

CHAPTER 4 – METROSCRIPT REFERENCE

APPENDIXES

Chapter 1
Getting Started

What is MetroScript?

MetroScript is a scripting language for use with MetroPro. It enables you to automate multiple and specialized operations with the click of a button. Examples of some operations are:

- Change control values or settings.
- Prompt the operator for input.
- Load a data file.
- Calculate and log special results.
- Combine multiple actions into one mouse click.

The key to MetroScript is writing a "script" file. The script file is a text file containing commands that MetroPro software executes.

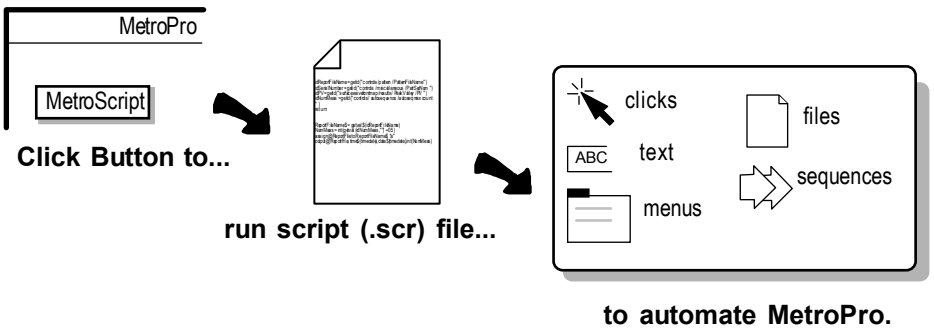
Scripts are particularly useful for operating an instrument in a production environment. Simple scripts can be created quickly. Yet, the language is powerful enough to enable an experienced programmer to write scripts to control complex measurement processes.

Requirements

You must have a licensed version of MetroPro to use MetroScript. It is included with MetroPro and is not a separate program.

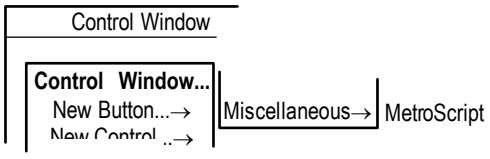
Using MetroScript

Once you have created a MetroScript file, you add a MetroScript button to a MetroPro application and link it to a script file.



The 5 Basic Steps

1. In MetroPro, create a MetroScript button in a Control window. Use the Control window menu and select New Button→ Miscellaneous→ MetroScript.

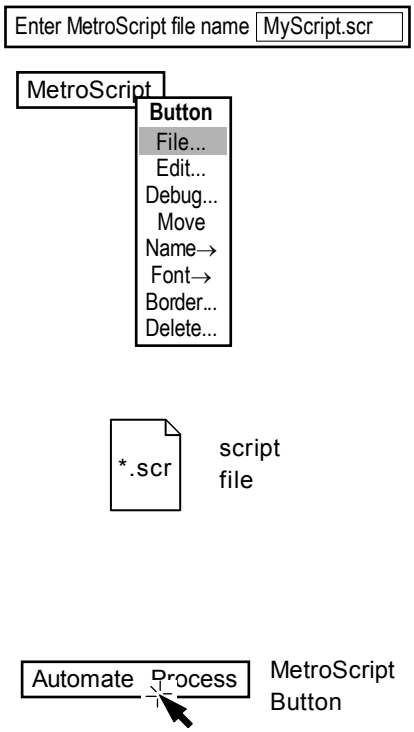


2. Specify the name of the script file to link to the MetroScript button. The file name should have ".scr" as the extension. Access the MetroScript button menu, select the File command, type the script file name, and press [Enter].

3. Create a script file using any standard text editor. You can run a text editor from MetroPro by accessing the MetroScript button menu and selecting the Edit command. Save the file to the MetroPro working directory.

4. Change the name of the MetroScript button to indicate its function. Access the MetroScript button menu, select the Name→ Rename command, enter a new name, and press [Enter].

5. Click the button to run the script.



Using this Manual…

This manual makes the following assumptions: 1) that you are already familiar with MetroPro, 2); that you are familiar with the computer's operating system; 3) that you have used the text editors included with the computer, and 4) that you have some understanding of programming concepts.

Familiarity with a BASIC language, such as Microsoft QuickBASIC, Microsoft Visual Basic, or a scripting language is recommended

Chapter2
MetroScript Fundamentals

Before You Begin

This chapter provides essential information about MetroScript language conventions. As with any other programming language, there are associated usage rules. This chapter covers these rules as well as important MetroScript command information that is interrelated with many MetroPro functions.

Terms Used in this Manual

The following terms are used in this manual. They are described in more detail in the following sections.

Term	*Description*
IntConst	An integer number.
RealConst	A decimal number, which may include an exponent.
HexConst	A hexadecimal number with 0x as a prefix.
StrConst	A sequence of characters within matching single or double quotes.
Const	Any *IntConst*, *RealConst*, *HexConst*, or *StrConst*.
RealVar	The name of a real number variable.
IntVar	The name of an integer variable.
StrVar	The name of a string variable.
Var	Any *RealVar*, *IntVar*, or *StrVar*.
FileVar	The name of a file variable.
LineNum	An integer from 1 to 99999999 used as a program line number.
LineNumRange	A range of line numbers.
LineLabel	A name used to label a program line.
LineId	Either *LineNum* or *LineLabel*.
SubstrRange	Specifies a substring in any of these forms: [start] [start, end] [start; length] For example: name$[3] color$[4,10]
IntExpr	An integer expression involving *IntVars* and operators.
RealExpr	An arithmetic expression involving *RealVars*, *RealConsts*, *IntVars*, operators and parenthesized *RealExpr*.

Term	Description
StrExpr	A string expression involving *StrVars*, *StrConsts*, concatenation operator (&), and parenthesized *StrExprs*.
LogicExpr	A logical expression involving two *RealExprs* or *StrExprs* compared by operators. Separated by operators And or OR. Parethesized *LogicExprs*.
Expr	Any *RealExpr*, *StrExpr* or *LogicExpr*.
FileNameExpr *FileNameConst*	A string specifying an existing file or a file to be created. *FileNameExpr* is a *StrExpr*. *FileNameCons*t is a *StrConst*.

Using Parentheses and Brackets

When writing constants, variables, or the expressions section of a MetroScript command, observe the following usage of parentheses and brackets. Parentheses ( ) are used for two purposes: 1) is for functions that take passed parameters—in this case the parameters are enclosed in parentheses; 2) is for enclosing array subscripts, for example, x(1) or y(2,3).

Brackets in italics *[ ]* are used to enclose optional items. Just remember, when writing scripts do not type the brackets when using options. Ellipses (. . .) appearing after a bracketed item denote repetition. Brackets are also used to enclose string subscripts. For example: name$[3].

Lexical Conventions

Components of a MetroScript file include commands, comments, and statements. Each component has associated usage rules, which are described in the following sections.

Commands

A load-time command is executed when a program is first loaded and is no longer visible when the program is run. A load-time command is not preceded by a tab or line number. Comments are not allowed on the same line as load-time commands.

The only useful load-time command is **loadsub**.

Comments

A comment is a section of code that is ignored by MetroScript when a program is executed. The exclamation point character (!) starts a comment at any location after the initial tab or line number. A '!' or **rem** (remark) starts a comment at the end of a statement. A comment continues until the end of the line.

Why use comments? Because having them in the code will help later on when the script has to be changed. Comments can be used to identify the function of the script, the name of the programmer, the date it was written, etc.

Statements

A statement is an expression that is preceded by a line number or tab and ends with a carriage return. An expression is a valid sequence of tokens. A token is the smallest piece of code that is recognized by MetroScript, and there are five types:

- Identifiers
- Keywords
- Constants
- Operators
- Commands

Spaces, tabs, and comments (collectively referred to as white space) are ignored except as they separate tokens. Indenting gives visual structure to the script. NotePad and WordPad do not provide auto indenting; tab characters at the start of a line must be inserted manually.

White space helps to separate items. Blank lines between logical sections of code help divide the script into smaller, workable sections. Spaces separate operators, operands, and pathname components, and make the code more readable.

Identifiers

Identifiers are sequences of letters, numbers, and underscores that are variable names or line labels.

They are case sensitive. My_String, MY_STRING, and my_string will all be recognized as unique names.

Variables are the programming convention used to store values in memory. These values are used when the program performs calculations and/or operations. Variables have a name, which is used to refer to the variable, and can be very helpful when debugging a program.

Variables also have a data type component. Data types identify the kind of data the variable can store. MetroScript data types are described in the section entitled, *Data Types and Variables*, later in this chapter.

Line Labels

Line labels are used with statements. Labels may contain upper and lower case letters and underscores, but no spaces. Labels must be contiguous.

Choose meaningful names for line labels. Those line labels which mark the start of a logical section of code (a subroutine), should start with a label that describes the function of the code, e.g., GetIDs, Init, GetSkewAngle, AnalyzeSite, etc. Line labels within a subroutine might also have descriptive names. One

convention is to have those names start with the same prefix. For example, within the subroutine MeasureSite, there might be line labels such as MS_Retry, MS_Error, etc.

```
gosub Measurement_Part1
on error goto No_Error
```

Note: Line labels and variable names may not duplicate MetroScript commands or functions. Remember that case makes a difference.

Keywords and Commands

The identifiers in the list below are reserved for use as keywords and may not be used otherwise.

abort	for	pause
assign	gosub	print
data	goto	printer
dim	if	read
disable	input	restore
else	is	return
enable	let	step
endif	logfile	stop
end	next	then
enter	off	to
enterline	on	wait
error	output	

Constants

A constant is a section of data that appears directly in the program. Constants are used to assign data to variables or to the print function.

Name and Description	Example
Integer Constants (*IntConst*) Any integer number.	3 0xfc3b
Real Constants (*RealConst*) Any decimal number, which may include an exponent.	2.3 –15.73e-6
String Constants (*StrConst*) Any sequence of characters within matching single or double quotes.	"Sample text" 'new program'

Special Characters in String Constants

There are special characters that can be used in string constants. A string constant used in a file name must follow certain path naming conventions. Refer to the following section.

Escape sequences are used to embed special characters in string constants. Some characters are not printable, and others are used to begin and end a string constant. Escape sequences begin with the backslash (\ ) character as shown below.

\b backspace
\f form feed
\n newline
\r carriage return
\t horizontal tab
\v vertical tab
**** backslash
\" double quote
\' single quote
\nnn octal character value (ASCII)

Path Names in String Constants

The string may be an **absolute path name** beginning with the forward slash character(/) or a drive letter followed by a semicolon (:), and containing any number of directory names. For example:

/temp/test.dat
C:/temp/test.dat

The string may be a **relative path name** beginning with a period and forward slash character(/) or the name of a directory within the current working directory and containing any number of directory names. For example:

./test.dat
run1/test.dat

The string may be a **simple file name** indicating that the file exists or will be created in the current working directory. For example:

test.dat

In MetroScript, the backward slash (\ )is interpreted as the beginning of an escape sequence to represent special characters in strings. Therefore, the forward slash (/)is recommended as a separator in absolute and relative path names.

Data Types and Variables

A variable is a named container for data. It is assigned a value from a constant or another variable. There are four data types in MetroScript: integer, real, string, and file. Special characters are used to indicate the type of variable. Keywords may not be used as variable names. All MetroScript variables are visible everywhere in the file in which they are defined.

Integer Variables (IntVar)

Integer variables start with an identifier and end with the percent character (%). They are also known as whole or counting numbers. Real data is truncated to an integer.

Assignment: `var_name% = 3`

Real Variables (RealVar)

There is no special character to indicate real variables.

Assignment: `var_name = 23.56`

String Variables

String variables end with the dollar sign character ($). They require a matched pair of either single or double quotation marks. The default size is 32 characters, but this may be changed by allocation.

Assignment: `var_name$ = "a string"`

Allocation: `dim var_name$ [ 256 ]`

Accessing:
```
var_name$ [ startNum ]
var_name$ [ startNum, endNum ]
var_name$ [ startNum; length ]
```

File Variables (FileVar)

File variables begin with the at character (@).

Assignment: `assign @identifier to fileNameString accessString`

Arrays

Arrays are ordered sequences of data. Arrays may contain any data type except file variables. Every element of an array must contain the same data type. Array indices are base one and therefore array dimensions must be greater than or equal to one. Array dimensions expressions are truncated to integer values, that is, they are rounded down to the next highest whole number.

The following examples show valid array statements.

```
dim b% (4,  3)
dim a$ ( 3 ) [ n% + 2 ]
dim c  ( len ( s$ )  * 2.5 )
```

Assignments

Assignment is the process by which a value is given to a variable. Assignment is by copy and not by reference. When one variable is assigned to another variable, as in a = b, it is done by copying the value of b to the variable, a. The following example shows a program that prints 1, not 2, as it might in other languages.

```
a% = 1
b% = a%
a% = 2
print b%
```

Operators and Expressions

Operators perform mathematical or logical operations on values. Operators are arithmetic, relational, or logical.

Arithmetic Operators *(ArithOpr)*		Relational Operators *(RelOpr)*	
Addition	**+**	Equal to (also assigment operator)	**=**
Subtraction	**-**	Less than	**<**
Multiplication	*****	Greater than	**>**
Divison	**/**	Not equal to	**< >**
Exponentiation	**^**	Less than or equal to	< =
		Greater than or equal to	**> =**
Logical Operators *(LogicOpr)*		**Miscellaneous Operators**	
Negation	**not**	String concatenation	**&**
Conjunction	**and**	Parentheses	**( )**
Disjunction	**or**	*Used for enclosing expressions, array subscripts, and function parameters.*	

DIV and MOD Operators

DIV returns the integer portion of the result of a division

Example:

-6 DIV 2 = -3
1 DIV 2 = 0
2 DIV 2 = 1
3 DIV 2 = 1
3.5 DIV 2 = 1
4 DIV 2 = 2

MOD returns the fraction portion of the result of a division

-1 MOD 3 = -1
0 MOD 3 = 0
1 MOD 3 = 1
2 MOD 3 = 2
3 MOD 3 = 1
4 MOD 3 = 1
5 MOD 3 = 2
5.5 MOD 3 = 2

Note: for either operator, the divisor cannot be zero (0). This will return a divide by zero error.

Operator Precedence

The operators are listed in order of highest to lowest precedence.

( )
^
not
-
&
*
/
+ -
= < >
<
>
< =
>=
and
or

Expressions

Expression components are defined as follows:

IntItem is defined as: `InConst` or,
`IntVar`

RealItem is defined as: `RealConst` or,
`RealVar`

StrItem is defined as: `StrConst` or,
`StrVar`

Integer Expressions (IntExpr)

This is an expression involving integer variables, integer constants, and the arithmetic operators. Integer expressions produce an integer value.

IntExpr is defined as: `IntItem` or,
`IntItem ArithOpr IntItem` or,
`( IntExpr )`

Real Expressions (*RealExpr*)

This is an expression involving real variables, real constants, integer variables, and the arithmetic operators. RealExpr is parenthesized. Real expressions produce a real value.

RealExpr is defined as:

```
IntItem or,
      Item or,
      Item ArithOpr  RealItem       or,
      RealItem  ArithOpr  IntItem   or,
      RealItem ArithOpr RealItem    or,
      (  RealExpr  )
```

String Expressions (*StrExpr*)

This is an expression involving string variables, string constants, and the concatenation operator. StrExpr is parenthesized. String expressions produce a string value. (When using a string expression in a path file name, refer to the *Constants*, "Path Names in String Constants" section of this chapter.

StrExpr is defined as:

```
StrItem           or,
StrItem & StrItem or,
(  StrExpr  )
```

Relational Expressions (RelExpr)

This is an expression involving two RealExprs or two StrExprs, and the relational operators. Relational expressions produce a truth value.

RelExpr is defined as:

```
RealExpr   RelOpr  RealExpr   or,
StrExpr  RelOpr  StrExpr      or,
( RelExpr )
```

Logical Expressions (LogicExpr)

This is an expression involving two *RealExprs* or *StrExprs* compared by operators. Logical expressions produce a truth value.

LogicExpr is defined as:

```
RelExpr     or,
      RelExpr  LogicOpr  RelExpr    or,
      (  LogicExpr  )
```

Flow Control

As with other programming languages, MetroScript includes the ability to control decision-making functions as well as the repetition of operations. The following section lists the statements that are used to manage flow control in a MetroScript program.

Decisions

```
if LogicExpr then Expression
if LogicExpr then Expression else Expression
if LogicExpr then
 Statements
endif
```

```
if LogicExpr then
 Statements
else
 Statements
endif
```

As shown below, **if** statements may be nested.

```
if LogicExpr then
 if LogicExpr then
  Statements
 endif
else
 if LogicExpr then
  Statements
 endif
endif
```

Repetition

```
for Variable = RealExpr to RealExpr
  Statements
next Variable
```

```
for Variable = RealExpr to RealExpr step RealExpr
  Statements
next Variable
```

Branching

goto lineId

gosub lineId

error return

return

on error gosub LineID

on error goto LineID

on RealExpr **gosub** LineID [, LineID] . . .

on RealExpr **goto** LineID [, LineID] . . .

As shown below, **for** statements may be nested.

```
for i = 1 to 10
  for j = 1 to 10
    Statements
  next j
next i
```

Other

end
stop
stop Expression

Subroutines

While MetroScript has a gosub keyword, it does not have user-defined subroutines (or functions) in the usual sense. A subroutine can be simulated with a label and a return statement. Data that would normally be returned from a function must be stored in a (global) variable.

To avoid clashes of variable names and labels, a naming convention can be very helpful. One such convention is to prefix labels and variable names with the name of the function that they are in. For example:

```
MyFunction:
  MyFunction_N% = 3
  goto MyFunction_End
  !  do something
MyFunction_End:
  ! do something
 MyFunction_Str$  = “some string”
  ! do something
return
```

Top Level and Application Scripts

Within MetroScript there is a differentiation between top-level and application-level scripts.

MetroPro scripts can be compared to macros used in a word processing program. A top-level script is similar to a macro that is defined outside of a *specific* document. An application-level script is similar to a macro that is defined *within* a specific document.

An **application-level** script has the following characteristics:
(Refer to the script samples in *Chapter 3* of this manual.)

- Has access to MetroPro objects defined within the application;
- Cannot access MetroPro objects outside the application (e.g. other applications);
- Can be run from a MetroScript button in the application;
- Can be run when the application is opened using Script controls.

A **top-level** script has the following characteristics:

- Has access to MetroPro applications;
- Has no access to MetroPro objects inside of applications;
- Cannot control a measurement;
- Can be run from the Desktop Menu option Script/Run;
- Can be run at start-up using the +script command line option.

The following functions can only be used in a top-level script:

- closeapp
- getappid
- loadapp
- openapp

Writing a MetroScript File

Starting a Script File

Use one of these methods to create a file.

- Outside of MetroPro, use any ASCII text editor to create a new script file.
- In MetroPro, create a new MetroScript button. Using the button pop-up menu, select **File** to specify a name for the script file. Select **Edit** to open the script file with a text editor so you can create and edit the file.

  Hint: It can be useful to associate the ".scr" extension with either the Notepad or Wordpad application.
- Access the Interactive Mode (see *Appendix B*).

Script File Location

Script files must be located in the MetroPro working directory in order for MetroPro to find them. If you start MetroPro from another directory, the script files must be located in that directory. The default working directory is: "\users\zygo".

Script File Format

Each program line should start with a tab character. This causes each program line to be automatically assigned a line number. This is the recommended programming style.

It is also possible to start each program line with a positive integer line number. Program lines are ordered by increasing line number. This style is not recommended, except for very short scripts because scripts with embedded line numbers are more difficult to revise.

Reminder End each script with a carriage return.

Program Structure

If the script is longer than what can be seen at one time in the text editor, break the script into logical sections, which perform specific functions, and call them with the **gosub** function. Not all program statements need to be put into subroutines; it may be appropriate instead to base the structure on logical grouping.

A program broken into subgroups would look like:

```
    gosub sub_A
    gosub sub_B
    end

!SUBROUTINES

sub_A:
    :
    :
    return

sub_B:
    :
    :
    return
```

MetroScript Commands and MetroPro Functionality

Many MetroPro functions can be simulated using MetroScript commands. The remaining sections of this chapter describe how to use commands for a desired function.

Measure, Acquire, and Analyze

MetroScript can perform the same functions as clicking the MEASURE button and the Analyze button in MetroPro. Unlike its MetroPro counterpart however, MetroScript can obtain unprocessed data with the **acquire** function. The following table summarizes the distinctions between the MetroScript **measure**, **acquire**, and **analyze** functions.

Function	Description
measure	Equivalent to clicking the MetroPro MEASURE button or pressing F1. For the Process window, it adds results to the process buffer if the Auto Store control is On. For the Report window, it logs or prints the report if the Logging or Printing controls are On.
acquire	Obtains data from the part under test, but no further processing is performed. It is the “scan” part of a measurement; it takes video data and generates "unprocessed" data; no results or plots are displayed. There is no MetroPro equivalent.
analyze	Equivalent to clicking the MetroPro Analyze button or pressing F2. It processes the “unprocessed” data according to Data window controls, and displays results and plots. When acquire and analyze are used together in sequence, it does not log reports, print reports, or add results to the process buffer. This must be done explicitly with the **logreports** and **storeprocstats** functions.

Accessing MetroPro Objects

An important aspect of MetroScript is the ability to access MetroPro objects. An object is any named item, for example, control buttons, attributes, plots, windows, and results. There are many reasons to do this, some of which are:

- Set control values to initial values at the start of a script.
- Change control values during a script for re-analyzing data.
- Read numeric results, controls or attributes for use in numeric calculations.
- Read control values to make decisions (e.g., if … then).

Object IDs

Each MetroPro object has a unique identifier or ID. MetroScript provides several functions for accessing MetroPro objects. Each of these functions requires knowing the object address. For most objects, this is done with the MetroScript **getid** function.

```
idObject = getid (MenuPathToObject)
```

The path for an object resembles the path for a file, except that instead of directories, the path contains menu items. The menu items are separated by the forward slash character (/). The path structure looks like:

WindowType / *MenuItem* / *SubMenuItem* / *SubMenuItem*

The window type is used for data windows. If no window type appears, then the path applies to Control windows. If the window is a Control window, do not enter "Control" in the path name.

Some path examples are:

```
" Controls / Acquisition / Phase Res "
" Attributes / Miscellaneous / Limits "
" Surface Wavefront Map / Results / Peak Valley / PV "
" Slope X Profile / Results / Rms "
```

It is often helpful to place all the **getid** calls in a section by themselves. This makes them easy to find and edit.

Hints for Typing Path Names

- Separate menu items by forward slash characters (/).
- Omit any slash characters that are embedded in Window names or menu items.
- The first menu item will always be “Controls,” “Results,” or “Attributes.” Note that the window type precedes the first menu item, except when it is a Control window; in which case the window type must not be entered.
- Path names are not case sensitive - use capital letters for readability.
- Spaces are allowed in path and file names.
- The returned address is an integer; there is no need to assign it to an integer variable.
- Data window types must be the full name, including "map" or "profile".

User-Defined Variable Names

User-defined variable names should be understandable. For example:

```
idPower = getid (" Surface Wavefront Map / Results / Power ")
idCameraMode = getid (" Controls / Interferometer / Camera Mode ")
idTestandRefDataWindow = getid (" Surface Wavefront Map(T+R) ")
```

Finding an Object's Path Name

To get the path for an object, there are three methods:

1. Access the item's menu and select the Name→ Identify command. A dialog box lists information, which will help you construct the path.
2. Point to the window in which the object is found and navigate the menus to find the object listing. The path contains the names of the items as they appear in the menus.

3. Print a settings file for the application type you are using. The settings file indicates the path structure to all controls and attributes in the application.

The Report Window and the getid Function

The getid function does not operate on Report window items. This is because Report windows are not unique. That is, two Report windows in the same application can have two different values of the same control.

Named Objects

It is also possible to get addresses for specific windows including Report windows and annotation boxes. This allows accessing the Window Control sub-menu functions for specific windows and writing information to specific annotation boxes. Addresses of this type are keyed to the object name, not the path.

To get a specific window address, use the **getwinid** function.

```
idWindow = getwinid (WindowName)
```

For example:

```
idSpecialDataWin = getwinid ("Special Data")
```

gets the address of a window which has been renamed "Special Data". To get a specific annotation box address, use the **getannotid** function.

```
idAnnotation = getannotid (idWindow , AnnotationName)
```

For example:

```
idSpecialResult = getannotid (idSpecialDataWin, "Special Result")
```

gets the address of an annotation box which has been *renamed* "Special Result" and which sits in the window identified by idSpecialDataWin. Because the **getannotid** function identifies a specific annotation in a specific window, you cannot use the **getid** function to get the window address which is fed to the **getannotid** function.

Identifying Masks

Masks are also identified for the purpose of manipulation. This type of address will be taken up in the "Controlling the Mask Editor" section.

Reading MetroPro Controls, Results, and Attributes

The getval and getval$ Functions

The whole purpose of identifying particular MetroPro objects is to either read their values or change their values. This is done with the **getval, getval$, setnum** and **setstr** functions.

RealVariable = **getval** (idObject, *UnitsString*)
StringVariable = **getval$** (idObject)

The numeric variable may be either an integer or a real. The string specifier indicates the units in which the value is to be given. This is simply the unit string as would be shown on the object itself. *It does not need to be the unit actually shown on the screen.* Examples of unit strings are:

Lateral	Linear	Height	Angle/Slope	Area	Volume
A	A	A	urad	sq A	nm3
nm	nm	nm	mrad	sq nm	um3
um	um	um	rad	sq um	
mm	mm	mm	sec	sq mm	
cm	cm	cm	min	sq cm	
m	m	m	deg	sq m	
uin	uin	uin		sq uin	
mil	mil	mil		sq mil	
in	in	in		sq in	
pix		wave			
		fr			
		z			

An object does not need to be visible on the screen for MetroScript to access its value with either of these functions. In other words, if you want to access the Camera Mode control, the control does not need to be present in the application.

A value with no units still needs a units specifier. In this case use the empty string, "".

Examples:

```
StepHeight = getval (idAvgHgt, "nm")
Power = getval (idPower, "wave")
Points = getval (idPoints, "")
Comments$ = getval$ (idComments)
```

List Controls/Attributes

For list type controls and attributes, either the **getval** or **getval$** functions can be used. Each selection of the list has an index, starting at 1. Each selection in the list has a string value. Either can be used.

Examples:

```
ScanLength = getval (idScanLength, "")
print ScanLength
   4

ScanLength$ = getval$ (idScanLength)
print ScanLength$
   40 um bipolar (25 sec)
```

When using **getval** for a list index, the units specifier is always the empty string.

Writing MetroPro Controls

It is often important to write to MetroPro controls. Typically this is done to initialize control values at the start of a script or to change control values if a script performs multiple analyses on the same data. Use the **setnum** or **setstr** functions for writing to MetroPro objects:

setnum (*idObject, RealValue, UnitsString*)
setstr (*idObject, StrExpr*)

Some examples:

```
setnum (idMinMod, 5, "")
setnum (idNomRailLength, 0.15, "mil")
```

```
setnum (idZernike8, SA3 / 6, "waves")
setnum (idScanLength, 3, "")
setstr (idComments, "This is a test")
setstr (idScanLength, "10 um bipolar (9 sec)")
setstr (idNamedAnnotation, val$ (Calculated Result))
```

Note that MetroScript can write to Control boxes and Annotation boxes, which have been named. It is not possible to write to Attributes; and Results are read only. The rules regarding the **getval** and **getval$** functions apply to the **setnum** and **setstr** functions as well.

Interacting with the Operator

Message Box

Message boxes are temporary displays of information. Their usefulness is limited. The **message** function displays a box with text in a pop-up window in the center of the screen.

message (*MessageString*, *MessageBoxCode, DisplayTime*,)

The message box code controls the color of the header bar, indicating the intent of the message:

Code	Color	Intent
1	green	informational
2	red	error
3	yellow	warning

Examples:

```
message (" hello world ", 1, 2)
message (msg$ = "The measurement failed ", 2, 5)
message (" The process took " & val$ (time$ (end_time - start_time)) & "
  seconds ", 1, 3)
message (" This is a long message \n It takes more than one line. \n This
  is the third line, ", 1, 5)
```

Hints for Using Message Boxes

- Message boxes are for temporary display only.
- Message boxes are always displayed at the center of the MetroPro window. If this covers important information, rearrange the application objects.
- The script execution does not proceed to the next line of code until the message box display is finished.
- Messages are easier to read if they include "blank-space". Some methods of using blank space are: 1) insert a space at the start and end of each line in the text, and 2) insert a line feed escape sequence - "\n" between lines of text or at the start and end of the message string.

Message Box alternative

Use a named annotation. This will not delay continuation of the script execution.

Dialog Box

The dialog box provides messaging capability, which requires operator acknowledgment. It can also provide branching input from the operator.

Response = **dialog** (*MessageString*, *DialogBoxCode*)

The dialog box code controls both the color of the header bar, indicating the intent of the message, and the buttons on the box:

Code	Color	Intent	Buttons
1	green	informational	OK button
2	red	error	OK button
3	yellow	warning	OK button
4	green	informational	Yes and No buttons
5	red	error	Yes and No buttons
6	yellow	warning	Yes and No buttons

Note: Pressing the Escape key in response to a dialog box generates a MetroScript error.

Examples:

```
ok = dialog (" Measurement Finished ", 1)

ok = dialog (" The part is out of spec: " & val$ (Calc Spec), 2)

ok = dialog (" Measurement Finished \n Click Yes to measure another
  part, Click No to stop ", 4)
if ok = 1 then goto NewPart
end

ok = dialog (" Measurement Failed \n Click Yes to try again,
  Click No to proceed to next part ", 5)
if ok = 1 then goto ReMeasure
goto NewPart
```

Hints for using message boxes apply to dialog boxes as well.

Annotations

Named Annotations provide a display which becomes part of the application. This has the advantage that it can be printed. Uses for Named Annotations include:

- Display of status messages, without halting execution of the script.
- Display of numeric results which can then be printed with the data (but not saved with the data set).

Example:

```
idStatWin = getwinid ("Status")
idStatBox = getannotid (idStatWin, "Status")
setstr (idStatBox, " Initializing parameters ")
```

Making a Status Annotation

The status annotation is very useful, both as a development tool and as a reassurance to the operator that all is proceeding as planned. To implement this tool:

1. Open (or create) the window in which you want the display to appear.
2. If the window does not have a unique name, rename it.
3. Create an annotation box in the window.
4. Rename the annotation box (from the menu).

In the script file:

1. In the GetIDs section of your script, add the lines:

   `idStatusWindow = getwinid` (*WindowName*)

   `idStatus = getannotid` (`idStatusWindow`, *AnnotationName*)

2. Every place in the code you want to display some information, insert the line:

   `setstr` (`idStatus`, *DisplayString*)

Status displays can include a variety of messages including:

Type of Message	Example String
Name sections of code	`"Initializing Parameters"`
Read values files	`val$ (int (NumMeas)) & " measurement locations"`
Timing data	`"Done" & val$ (EndTime - StartTime)`
Error messages	`val$ (int (errn)) & "  " & errm$`
Calculated values not for final display	`"New stage position: " & val$ (x) & val$ (y)`

Control Boxes

Control boxes can also be used to display information. Though this is not so useful for strings, the formatting of the control box (units, value format, and layout) can be used to advantage.

Accepting Parameters at Run Time

Often a script will require some information that is not available until runtime. For example, the number of measurements to make, a serial number, the name of an output file, etc. There are several methods of reading parameters at run-time.

The promptstr$ Function

This **promptstr$** function creates a pop-up box to solicit input. An example:

promptstr$ (*PromptString*, *DefaultString*, *MaxInputLength*)

The box appears in the middle of the screen and does not disappear until the enter key or the escape key is pressed, or a mouse button is clicked.

If the entry is supposed to be a numeric value, check the validity of the entry before proceeding.

Examples:

```
Comment$ = prompstr$ ("Enter comment here", "No comment", 80)

NumberEntry:
   temp$ = prompstr$ ("Type the number of measurements", "4", 3)
   on error goto InvalidNumberString
   NumMeas = val (temp$)
   off error
         :
         :
InvalidNumberString:
   off error
   ok = dialog (" \n Invalid number \n \n Please try again \n ", 3)
   goto NumberEntry
```

Unless there is a compelling reason to prompt the user, it is recommended that control boxes be used for entering numeric values.

In the case of the Lot Num, Part Ser Num, and Auto Save Data File controls see the **promptentry** function in Chapter 4.

Input from Control Boxes

MetroPro controls provide a good way to enter parameters. In addition to using the standard controls for the application, it is possible to "commandeer" unused controls. That is, take an otherwise unused control and use it for a parameter which is needed by the script.

Example:

```
idMultiplier = getid ("Controls / Miscellaneous / Part Thickness ")
Multiplier = getval (idMultiplier, "nm")
```

In this example, the control for Part Thickness has been used for another purpose. In the application, you would create the control and rename it to "Multiplier". In this way the operator could click on the Multiplier control, type in a value, and proceed as if Multiplier were a standard MetroPro control.

Finding Controls to Take Over

For a listing of available controls, see *Appendix C*.

There are many controls available in a Control window. In data windows, the available controls are significantly fewer. It is essential that you select a control that will have no unwanted effect on any part of the application use.

If making a script for a GPI, select controls normally used for the microscopes. If making a script for the microscope, select controls normally used for the GPI.

> ***Hint:*** Avoid using the Zernike Coefficient controls if: either the Wavelength-In or Wavelength-Out controls are changed, or the application is to be copied to other systems. These controls interact with one another; values can change.

Controls that toggle between On and Off values are useful for controlling branching in the script. These controls can be set to display On/Off, Yes/No, or True/False by selecting the Display As command from the control's menu.

Controls that are used for integers should be truncated to the nearest integer when read with the **getval** function. MetroScript contains no "nint" function, so this truncation takes the form of:

```
idNumMeas = getid (" Controls / Autosequence / Auto Seq Max Count ")
NumMeas = int (getval (idNumMeas, "") + 0.5)
```

Here the control is a numeric control that has no units.

Hints for Using Control Boxes

- Controls can be renamed as necessary. There is no need to economize on the length of the name. Its purpose should be clear.
- Text controls can be used for a number of purposes in addition to storing comments. Most significant among these is to store a file name for some ancillary file which the script is either to read or to write.
- When possible, use a unitless control for a unitless parameter. If the parameter has units, it is best if a control with the same type of units can be found. If this is not possible, then it is recommended to change the Layout to "Name Value" to avoid confusion.
- Set the Value Format command appropriately for the parameter.
- If the value is an integer, add 0.5 to it and truncate (see the **getval** function).

Interacting With MetroPro Files

MetroScript provides for reading and writing the standard MetroPro files (data, masks, patterns, settings), except Report files. It also allows for writing sequential ASCII files to the disk. These may be used to create and/or read:

- customized report files
- stitching coordinate files (*.crd)
- stitching sequence files (*.seq)
- measurement location files (*.sit) (sites)

This type of I/O uses the following functions:

assign *@FileVar* **to** *FilenameString AccessString*

The file variable name begins with an @ sign. The file name string may include a path. The access string parameters are shown in the following table.

Parameter	Function
"w"	open for writing - erases existing file of this name.
"r"	open for reading only.
"a"	append - opens for writing but does not erase existing data.

To close the file, use the assign function with the following syntax,

```
assign @FileVar to ""
```

Reading from and writing to the file are accomplished with the **enter** and **output** functions. Each **enter** function corresponds to one line in the input. A semi-colon at the end of an **output** statement suppresses the linefeed at the end of the line.

```
enter @FileVar; variable_1, variable_2, …
output @FileVar; variable_1, variable_2, …
output @FileVar; variable_1; variable_2; …
```

Examples:

```
dim SpecialResult(10)
   :
   :
assign @ReportFile to "process.rep" "a"
output @ReportFile; "      index           Result"
for i = 1 to n
   output @ReportFile; int (i), SpecialResult(i)
next i
assign @ReportFile to ""

dim x(10)
dim y(10)
   :
   :
assign @CoordFile to "xycoords.sit" "r"
for i = 1 to n
   enter @CoordFile; x(i), y(i)
next i
assign @CoordFile to ""
```

Formatting Output of Numbers

The output of numbers is governed by the integer and real number formatting functions: **ofmtr** and **ofmti**. The functions take a string argument which follows formatting conventions used by the C language "printf" function.

Example	Description
`ofmtr ("%10.4f")`	Real numbers displayed in floating point format; field of 10 spaces with four digits to the right of the decimal point.
`ofmtr ("%.2f")`	Real numbers with two decimal places with no leading spaces.
`ofmtr ("%08.4f")`	Field of eight spaces, forces leading zeroes.
`ofmtr ("")`	Return to default real number formatting.
`ofmti ("%5d")`	Integer formatting in a field of five spaces.

To use the integer formatting, either use integer variables (with % as last character in name) or use the **int** function on the real variable when outputting it.

In the **output** statement, variables may be separated by commas or semicolons. The comma indicates that the output should be spaced in standard size fields. Use of the semi-colon suppresses any additional spaces beyond those specified in the **ofmtr** and **ofmti** functions.

Examples:

```
ofmti ("%8d")
ofmtr ("%8.3f")
output @FileVar; n%, x, y, s$
```

```
14    2.345   -0.875    hello world
```

```
output @FileVar; n%; x; y; s$
```

```
14    2.345   -0.875hello world
```

```
for i = 1 to 3
      output @FileVar; x(i)
next i
```

```
   1.234
   6.789
  -7.654
```

```
for i = 1 to 3
      output @FileVar; x(i);
next i
```

```
 1.234    6.789    -7.654
```

Controlling the Mask Editor

MetroScript provides limited interaction with the MetroPro Mask Editor.

Function	Description
adjustmask	Changes the mask width and height.
loadmasks *(FileNameExpr)*	Loads the specified mask file.
movemask	Moves mask to specified location.
savemasks *(FileNameExpr)*	Saves the current mask with the specified file name.
resetmasks	The equivalent of clicking on the Clear button in the Mask Editor.

Mask manipulation is done with the following functions:

idMask **= getmaskid** (*XCtrCoord*, *YCtrCoord*, *UnitsString*)

The **getmaskid** function returns the id of the mask figure for which the minimum enclosing rectangle has its center closest to the coordinates specified in the function call. The coordinates are based with 0,0 as the center of the camera, except when using lateral units of pixels ("pix") for which the origin is the lower left corner of the camera.

movemask (idMask, *NewXCtrCoord*, *NewYCtrCoord*, *UnitsString*)

The **movemask** function moves the center of the minimum enclosing rectangle of the specified figure to the coordinates specified in the call. It follows the same units/origin convention as getmaskid.

adjustmask (idMask, *NewWidth*, *NewHeight*, *UnitsString*)

The **adjustmask** function applies to the minimum enclosing rectangle of the figure specified in the function call. It follows the same units/origin convention as **getmaskid**.

For the **movemask** and **adjustmask** functions, when idMask = 0, all mask figures are affected; it is equivalent to a "pick all" in the Mask Editor.

Remote Control of MetroPro

MetroScript provides for remote control of MetroPro through an RS-232 interface.

"COM" Ports

PC compatible computers used with ZYGO instruments are provided with two serial or "COM" ports. The default designation of most computers built-in serial ports are COM1 and COM2. COM1, COM2, COM3, and COM4 ports may be used for remote control purposes.

Four COM port settings can be set. These settings are: baud rate, bits, parity, and stop bits. Use the assign statement to set values. For example, to set COM1 to 57600 baud rate, 8 bits, odd parity and 1 stop bit, use the statement as shown below. Note that values must be set in the following order: baud rate, bits, parity, stop bits.

```
assign @Sio to "com1" "57600, 8,1,1"
```

However, if using the default setting, which is 9600 baud, 8 bit, no parity, and 1 stop bit, use the following: `assign @Sio to "com1."`

Value Range for Each Setting

Baud Rate	Bits (per byte)	Parity	Stop Bits
150	5	0 no parity	1
300	6	1 odd parity	1.5
600	7	2 even parity	2
1200	8		
2400			
4800			
9600			
19200			
38400			
57600			
115200			

The PC serial ports use 9-pin D-subminiature connectors with RS-232 standard Data Terminal Equipment (DTE) signals. Most remote control configurations require only the RXD, TXD, and GND signals.

Pin No.	Signal	Description
1	DCD	Data carrier detect
2	SIN	Serial input (RXD)
3	SOUT	Serial output (TXD)
4	DTR	Data Terminal Ready
5	GND	Ground
6	DSR	Data Set Ready
7	RTS	Request To Send
8	CTS	Clear To Send
9	RI	Ring Indicator

Input and Output

The serial port device file is opened like an ordinary file using the **assign** statement as shown below.

```
assign @Sio to "com2"   ! PC compatible example
```

Characters are transmitted using the **output** command as shown in the following example.

```
dim TxData$[100]
TxData$ = "hello world"
gosub Tx
stop
Tx:
output @Sio; TxData$
return
```

The default output terminator is an ASCII new-line character (hex 0A). Characters are received using the **enter** command as shown in the following example:

```
        dim RxData$[100]
        gosub Rx
        print RxData$
        stop

   Rx:
        on error goto RxErr
   Rx2:
        enter @Sio; RxData$
        off error
        return
   RxErr:
        if pos( errm$, "Ran out of input" ) <> 0 then goto Rx2
        print errm$
        stop
```

The **on error** is needed because the **enter** statement will result in a "Ran out of input" error if a new-line terminated string is not received within two seconds.

Remote Abort

It may be desirable to allow the remote controller to abort the current processing. To support this capability there is an **enable abort** MetroScript command, which is used like this:

```
enable abort @Sio
```

Then, if an ASCII escape character (hex 1B) is received (while not executing **enter @Sio**), it will result in a "Processing aborted" error. (The effect is as if the operator pressed the Esc key on the keyboard.) This error can be handled using the **on error** mechanism.

To be thorough, after an **enter @Sio** statement, the received string should be checked to see if it contains an escape character as in this example:

```
if pos( RxData$, chr$(27) ) <> 0 then goto RxEsc
```

The ability to handle the "Processing aborted" error using the **on error** mechanism is new with MetroPro 6.5.7. Previously, this error always caused a script to be aborted. To ensure that a poorly designed script using the **on error** mechanism cannot hang up MetroPro, the **on error** mechanism will fail if the escape key is pressed two or more times.

Note the use of the **enable abort** should only be in areas where received data is not expected (e.g. during measurements, execution of patterns, or other lengthy operations).

The **enable abort** command under certain circumstances can read and discard an incoming character, thereby not allowing it to be seen by an enter command. It is recommended that enable abort be used in conjunction with **disable abort** as shown in the following example:

```
Meas_It:
     on error gosub Chk_Meas_Abort
     enable abort @Sio     ! enable remote abort
     measure
     disable abort   @Sio! disable so we don't lose any characters
     return
Chk_Meas_Abort:
     if pos( errm$, "Processing aborted" ) <> 0 then
        ok = dialog(" Measurement Aborted ", 1)
     else
        ok = dialog(" Measurement Error Occurred ", 2)
     endif
     error return
```

Considerations for Motion Control and Joystick Usage

MetroScript is often used to control stage motion with the Nova Motion Controller. In particular, managing the state of the joysticks requires careful attention when developing scripts. This section provides information about the issues related to the Nova Motion Controller as they relate to creating well-behaved scripts.

Motion Controller Basics

When using the motion controller with MetroScript commands, observe the following principles. MetroPro and MetroScript try to manage commands to the controller to avoid violating these principles.

- A joystick canot be disabled while the operator is using it.
- A programmed move command is rejected if the joystick is enabled.
- A joystick cannot be enabled during a programmed move.

Script Design Decisions

There are two major design decisions to consider when writing scripts to manage the joysticks. They are: concurrent motion and error handling.

Ignore the Joysticks—A script that simply ignores the joysticks is the easiest to write. However, it cannot be robust. For example, if the joysticks are enabled and the operator moves the joystick just prior to the script executing a programmed move, the move command will generate an error.

Disable the Joysticks—A script that disables the joysticks during execution is practical if the operator does not need to use the joysticks during the run. The script should use error handling to ensure that the joysticks are always re-enabled before exiting. The programmer must be aware of commands that may unexpectedly re-enable the joysticks.

Enable the Joysticks Intermittently—A script that enables the joysticks intermittently is required if the operator must use the joysticks during the run.

Concurrency—A script that perfoms stage movements concurrent with other operations may be necessary to meet performance goals. In this situation, the “Don’t-Wait-For-Stop” move commands must be used.

Error Handling—This can be the most difficult aspect of writing a production-quality script. Encountering both process errors, as well as anticipating operator interaction, (including activity such as repeatedly pressing the Esc key), poses multiple challenges to writing scripts that include robust error handling routines. Managing conditions such as concurrent motion and the state of joysticks adds more complexity. It is possible for no error handling to be better than poorly designed error handling as occasionally the handling routine will produce results that are difficult to explain.

When writing scripts that will include error handling, keep the following in mind:

- Assume that an error can be generated at any time by virtue of the operator pressing the Esc key at any time.
- Always have an **on error goto** or **on error gosub** in effect.

- The error handler should test the value of **errm$** to find out which error has occurred. For example, it should not simply assume that if **measure** failed, the part must be out of focus. As a minimum, the error handler should test for the string "Procesing aborted" to find out if the operator is trying to stop the script with the Esc key.
- The error handler may be re-entered as a result of the operator pressing the escape key at just the "wrong"time.
- The error handler should provide optional logging of error messages as a diagnostic.

Motion Related MetroScript Commands

The following table lists the primary MetroScript stage movement commands, segregated into "Wait-For-Stop" and "Don't-Wait-For-Stop" commands.

Wait-For-Stop Commands	Don't-Wait-For-Stop Commands
movexy(X,Y) movexy(X,Y,1)	movexy(X,Y,0)
movexyz(X,Y,Z) movexyz(X,Y,Z,1)	movexyz(X,Y,Z,0)
movez(Z) move(Z,1)	movez(Z,0)
moverp(R,P) moverp(R,P,1)	move(R,P,0)
move(T) movet(T,1)	move(T,0)
moveaxis("X",0,X,1)	moveaxis("X",0X,0)

The wait-for-stop forms of the move commands are the easiest to use. They perform the following sequence:

1. Record the initial state of the joysticks.
2. Disable the joysticks if they were enabled.
3. Command the Motion Controller to start moving.
4. Wait for the Motion Controller to finish moving.
5. Restore the joysticks to the initial state.

The don't-wait-for-stop forms of the move commands allow for concurrent operations.

The commands perform the following sequence:

1. Disable the joysticks.
2. Command the Motion Controller to start moving.

The script must wait for movement to complete, and manage re-enabling the joysticks. If the script tries to enable the joysticks before all movement is completed, the **joyon** command generates an error.

As of MetroPro version 8.1.5, the Don't-Wait-For-Stop form of the **moveaxis** command tries to restore the joysticks to the intial state. If the joysticks were previously enabled and the move is non-zero, the following diagnostic message is output to the log file.

```
mc_enable_joy() failed – ERR_MC_JOYSTICK_ENABLE
mc_pop_joystick() failed – ERR_MC_JOYSTICK_ENABLE
```

Note that the command does not generate a MetroScript error. To avoid the diagnostic message, ensure that the joysticks are disabled before using this form of the command.

There are several MetroScript commands that perform pattern-related stage movements, including:

gotopatorg gotopatload gotopatpos

These commands perform the following sequence:

1. Record the initial state of the joysticks.
2. Disable the joysticks if they were enabled.
3. Command the motion controller to start moving.
4. Wait for the motion controller to finish moving.
5. Restore the joysticks to the intial state.

All of the move commands operate as follows: If the joysticks were previously enabled and the operator was moving an axis, the command will fail when it tries to disable the joysticks.

Joyon and Joyoff Commands

The **joyon** command generates an error if a programmed move is in progress. The **joyoff** command generates an error if a joystick is in use.

Wait-For-Stop-Commands

There are several MetroScript commands that perform a wait-for-stop operation. They are useful after a Don't Wait-For-Stop Move command is used. The commands are:

waitxy waitxyz waitz waitrp waitt

Timing/Communication Failures

A simple script, as shown below, should have worked perfectly. But, sometimes one or more locations were omitted from the stitch.

```
loadpattern("MyPattern.pat")
gotopatorg
runscript(0, "UserStitch.scr")
```

When that happened, the MetroPro log file contained the following message:

```
mc_enable_joy() failed - ERR_MC_JOYSTICK_ENABLE
mc_pop-Joystick() failed - ERR_MC_JOYSTICK_ENABLE
```

The problem was fixed by inserting a 50 millisecond delay into a low-level joystick control function.

Script Compatibility

While scripts are generally compatible between systems and versions of MetroPro, you may encounter problems under the following conditions:

- a button or control was moved to a submenu,
- a directory name was changed,
- a directory was moved or deleted,
- some hardware changes.

Before upgrading to a new version of MetroPro/MetroScript, review existing scripts to determine if any of these conditions have occurred.

Chapter 3
Example Scripts

Note: The first two examples illustrate in detail all five steps as mentioned in *Chapter 1*. The remaining examples in this chapter show only the sample script file. To implement a script, all five steps should be performed.

Be sure to use the tab key when writing script files. All active script files should be stored in the working directory.

Reminder: End each script with a carriage return.

Example 1 - Perform a Function

This example creates a button that clears masks from the application; it reduces four mouse clicks to one.

1. In MetroPro, create a MetroScript button. Use the Control window menu and select New Button→ Miscellaneous→ MetroScript.
2. Specify the name of the script file to link to the MetroScript button. Access the MetroScript button menu, select the File command, enter the script file name, and press [enter]. In this example link the button to a file named "ClearMask.scr".
3. Create the script file. Access the MetroScript button menu and select the Edit command. In the text editor type the following; [tab] and [enter] refer to keys on the keyboard.

   [tab] `resetmasks` [enter]

   Save the script file to the working directory and then exit the text editor. In this example name the file "ClearMask.scr".
4. Change the name of the MetroScript button to match its function. Access the MetroScript button menu, select the Name→ Rename command, enter a new name, and press [enter]. In this example name the button "Clear Mask".
5. Test the script file. Open the Mask Editor and create a figure. Click the Clear Mask button to perform the script; the figure should be deleted.

Example 2 - Enter a Control's Value

This example changes the value of the Wavelength Out MetroPro control with a preset value and units so that the GPI scales results as if the test was made at another wavelength.

1. In MetroPro, create a MetroScript button. Use the Control window menu and select New Button→ Miscellaneous→ MetroScript.
   In addition, select New Control→ Master Units→ Wavelength-Out to create the Wavelength-Out control. Using the control's menu, set the Layout to Value Units and the Value Format to 0 decimals and 4 width.
2. Specify the name of the script file to link to the MetroScript button. Access the MetroScript button menu, select the File command, enter the script file name, and press [enter]. In this example link the button to a file named "Set488.scr".
3. Create the script file. Access the MetroScript button menu and select the Edit command. In the text editor type the following: [tab] and [enter] refer to keys on the keyboard.

   [tab] `idWLOut = getid ("Controls/Masterunits/Wavelength-out")` [enter]
   [tab] `setnum (idWLOut, 488, "nm")` [enter]

   Save the script file to the working directory and then exit the text editor. In this example name the file "Set488.scr".
4. Change the name of MetroScript button to match its function. Access the MetroScript button menu, select the Name→ Rename command, enter a new name, and press [enter]. In this example name the button "Set 488".
5. Test the script file by clicking the Set 488 button. Verify that the value of the control has changed.
6. To automatically load the script file when the application is loaded, perform the following steps: Create the Auto Run Script and Script File controls in a Control window. Set the Auto Run Script control to On and enter the name of the file (Set488.scr) in the Script File control. Save the application.

Example 3 - Select a Control's Setting

This example script file sets a control with multiple selections to a predetermined setting. By adding additional lines, the same script could be used to select the settings of numerous controls at the same time.

```
idCameraMode = getid ("Controls / Interferometer / Camera Mode")
setstr (idCameraMode, "640x480")
```

Example 4 - Writing Results to a File

This script appends header and result information to a report file named "Results.rep" when the button is clicked.

```
ofmtr ("%8.3f")  !set output formats
ofmti ("%8d")
assign @ReportFile to "Results.rep" "a"  !open file to append
t =timedate
output @ReportFile; ID$, date$(t), time$(t)
for i =1 to n   !loop through results
   output @ReportFile; int (i), Ra (i)  !write meas. num. and value
next i
assign @ReportFile to ""  !close file
```

Example 5 - Get Results for Specific Regions on a Part

This script file locates two distinct areas on the part, measures PV in each location, displays both values in a dialog box and asks the operator for input. The distinct areas or regions must be previously defined with user-created masks.

```
idPV = getid ("Surface Wavefront Map/ Results/ Peak Valley/ PV")
loadmasks ("mask1.mas")
analyze
PV1 = getval (idPV, "waves")
loadmasks ("mask2.mas")
analyze
PV2 = getval (idPV, "waves")
msg$ = "PV1:" & val$ (PV1) & "\n PV2" & val$ (PV2)
ok = dialog (msg$, 1)
```

Example 6 - Configure MetroPro for Multiple Parts

This example uses the MetroPro settings file to quickly and easily change the settings of numerous controls within the application. Create a MetroScript button for each type of part you want to measure.

Before running the script file, the settings file should already exist. To make a settings file, configure the software controls for measuring the part and click the Save Settings button, enter a name for the file ending with ".set", and press [enter]. The Save Settings button is available in a Control window; it is created with the New Button→ command.

```
loadsettings ("part1.set")
measure

loadsettings ("part2.set")
measure
```

Example 7 - Automate a Measurement Process

This script file automates multiple measurements of the same part. A mask file is loaded to select an area of the part to measure; the measurements are made; the results are stored in the Process Stats, and then saved to a file.

```
dim LotName$[200]
dim PartName$[200]
dim MaskName$[200]
dim MeasFn$[200]
dim ErrMsg$[200]

SAMPLE_PARTS:
     Exit_Now = 0

     on error gosub MASK_LOAD_ERR

     NumSamplesId = getid( "Controls/Stitch/NCols")
     NumSamples = getval( NumSamplesId, "")
     LotNameId = getid( "Controls/Miscellaneous/LotNum")
     LotName$ = getval$( LotNameId )
     PartNameId = getid( "Controls/Miscellaneous/PartNum")
     PartName$ = getval$( PartNameId )

     MaskName$ = LotName$ & ".mas"
     resetmasks
     loadmasks( MaskName$ )
     clearprocstats
     for x = 1 to NumSamples
      on error gosub MEAS_ERR
      measure
      storeprocstats
      MeasFn$ = LotName$ & "_" & PartName$ & "_" & val$(x) & ".dat"
      savedata( 0, MeasFn$ )
     next x
     saveprocstats( LotName$ & "_stat.tbl",0 )
     goto SAMPLE_PARTS_EXIT

MEAS_ERR:
     off error
     ErrMsg$ = " Error in measurement " & val$(x) & ", Continue? "
     ok = dialog( ErrMsg$, 4 )
     if ( ok = 0 ) then goto SAMPLE_PARTS_EXIT
     error return

MASK_LOAD_ERR:
     off error
     ErrMsg$ = " Error loading mask '" & MaskName$ & "', Continue? "
     ok = dialog( ErrMsg$, 4 )
     if ( ok = 0 ) then goto SAMPLE_PARTS_EXIT
     error return

SAMPLE_PARTS_EXIT:
     end
```

Example 8 - Handling Errors

The first example shown below is a generic error script; the second script is specific to a microscope with programmable stages.

```
   On error gosub SetError_Flag
   .
   .
ReMeasure:
   measure
   if Err=1 then
      Err=0
      OK = dialog ("The following error occurred during
      measurement" \n & errm$ & "\n Click Yes to retry,
      click No to quit", 5)
      if OK = 1 then goto ReMeasure
      end
   end if
      .
      .
SetError_Flag:
   Err=1
   return
   on error gosub Move_Error
   movexy (xpos, ypos)
   off error
   .
   .
Move_Error:
   if pos (errm$, "out of range") >0 then
      ok = dialog ("Stage position is out of range"& val$ (xpos)
      & val$ (ypos), 2)
      end
   end if
   if pos (errm$, "communicating")>0 then
      ok = dialog (errm$ & "\n Click Yes to retry,
      click No to quit", 6)
      if ok=1 then error return
      end
   end if
   if pos (errm$, "host")>0 then
      ok = dialog ("Please let go of the joystick", 3)
      resetmc
      return
      end
   end if
   ok = dialog (errm$ & "When attempting to move to position" &val$ (xpos)
    & val$(ypos) & "\n halting script", 2)
   end
```

Example 9 - Set a Result Value

This example script file sets the value of a result box. This is particulary useful for setting the values in a Custom result box.

```
// Get the id of two standard MetroPro results

idTiltXTest = getid ("Surface Wavefront Map/ Results/ Tilt/ Tilt X")
idTiltXTest2 = getid ("Surface Wavefront Map 2/ Results/ Tilt/ Tilt X")

// Get the values of these results

tiltXTestVal = getval( idTiltXTest, "degrees")
tiltXTest2Val = getval( idTiltXTest2, "degrees")

// calculate the angle between the two surfaces

angle = tiltXTest – tiltXTest2

// get the id and set the value of the custom result

idAngleX = getid("Surface Wavefront Map 2/ Results/ Custom/ Custom Result
1")

setnum(idAngleX, angle, "degrees")
```

Example 10 - Set Position Status

Generally, a Stage Control Pattern is set up to make a measurement at each pattern position. In this case, the Position Status Map displays different letters to indicate the result at each position.

A less typical use of the Stage Control Pattern function is when it is set up to run a script at each pattern position. This is done by setting the Operation field to Run Script and specifying the script filename. In this case, the script is responsible for setting the pattern position status. If the script does not set the position status and completes normally, the default position status is "P".

The **setpatposstat** function can be used to allow a script to set a position status. The following sample script shows usage of this function.

```
acquire
analyze
id = getid( "Attributes/ Pattern / Pattern Row" )
Row =  getval ( id, "" )
id = getid( "Attributes / Pattern / Pattern Col" )
Col = getval (id, "" )
id = getid( "Attributes / Miscellaneous / PassFail" )
Pass = getval( id, " " )
if Pass = 1 then
 Status$ = "P"
else
 Status$ = "F"
endif
setpatposstat( Row, Col, Status$ )
```

The status letter normally must be one of the following. Case is not significant.

Character	Color	Status
P	Green	Pass
F	Red	Fail
?	Yellow	Could not measure
#	Green	No operation
O	Red	Out-of-Limits
A	Yellow	Abort

Note: The sequence "acquire" and "analyze" are used instead of "measure." The "measure" command would reset the Pattern Row and Pattern Col attributes therefore defeating the above logic.

Error Handling

Pattern Script Fatal Error

A pattern script fatal error can occur if 1) the script could not be run, either because the file was not found or it contained syntax errors; 2) the script ran but generated an error that that was not handled via the “on error” mechanism.

A pattern script fatal error always sets the position status to yellow “?” and causes the pattern to abort. An error dialog is displayed (the Failure Action control is ignored).

Stop Value

The script can use the stop command with a numeric expression parameter (stop value) to achieve some rudimentary control over error handling and the position status value. For example: stop 1

If the stop value is zero, it is the same as using the stop command without a parameter. The Position Status is set to green “P” for pass unless the **setpatposstat** was set to another value (see the table following the example script). The pattern continues normally.

If the stop value is negative one (-1), then the position status is not changed from the initial white “I” unless the setpatposstat function set another value. The pattern is aborted and an error message is displayed (the Failure Action control is ignored).

If the stop value is any other value, then the position status is always set to yellow “?” (over-riding any use of setpatposstat). A pattern error is generated and handled according to the Failure Action mechanism.

Chapter 4
MetroScript Reference

Reference Dictionary

This section describes MetroScript functions and statements; they are listed in alphabetical order. These functions and statements are best used in a program. Some of them may be used in Interactive (Debug) mode, but refer to *Appendix B* before doing so. Return values are type real unless the function name ends with "$", in which case the return value is type string. Conversion between integer and real values is automatic where required.

Note: Statements *cannot* be used in an expression.

RealVar = RealExpr

This assignment statement evaluates the *RealExpr* and assigns the result to the variable.

IntVar = RealExpr

This assignment statement evaluates the *RealExpr,* truncates any fractional part, and assigns the result to the variable.

StrVar = StrExpr

This assignment statement evaluates the *StrExpr*, and assigns the result to the variable.

abs(*RealExpr*)

Returns the absolute value of *RealExpr* parameter.

accumdata(*FileNameExpr*)

acquire

Causes MetroPro to acquire data from the instrument without analysis. There is no equivalent in MetroPro. Returns no value.

acs*(RealExpr)*

Returns the arc-cosine of the value of *RealExpr* in the current angle unit.

adddata*(FileNameExpr)*

Causes MetroPro to add to the current data the data file specified by *FileNameExpr.* Returns no value. Equivalent to clicking on the MetroPro Add Data button and entering the data file name.

adjustmask*(IntExpr, RealExpr1, RealExpr2, StrExpr)*

This function changes the dimensions of a mask figure; it does not return a value.

The *IntExpr* parameter specifies the ID of a mask figure obtained via **getmaskid**. The new X and Y dimensions are specified by *RealExpr1* and *RealExpr2*, respectively. The measurement unit is specified by *StrExpr*. If null, the default unit is pixels. If either dimension is zero, the mask figure is deleted.

The dimensions of the coordinate system are as described for the **getcamsizx** and **getcamsizy** functions.

analyze

Causes MetroPro to reanalyze the current data. Equivalent to clicking on the MetroPro Analyze button.Returns no value.

analyzeftpsi(*IntExpr*)

This command generates a phase map by performing the MST analysis. There must be data available to perform the analysis. *IntExpr* specifies the source of the MST data. For all MST applications besides Homogeneity, the source must be zero (0). When loading data into the MST Homogeneity application, set *IntExpr* to zero (0)for full cavity, and one for empty cavity data. Data may be loaded using the MetroScript command, **loadftpsi**; manually loaded by clicking the Load Data button within an Acquisition Data window, or from the most recent measurement.

This function is available in the following MST applications: Two Surface, Parallel Surface, Homogeneity, and Custom Cavity.

analyzeftpsipeak*(IntExpr, RealExpr, StrExpr)*

This command generates a phase map by performing MST analysis. The analysis occurs at a peak chosen by the *RealExpr* parameter, corresponding to the one-pass optical path length between two parallel surfaces. *StrExpr* specifies the units. There must be data available to perform the analysis. *IntExpr* specifies the source of the MST data. For all MST applications besides Homogeneity, the source must be zero (0). When loading data into the Homogeneity application, set *IntExpr* to zero (0) for full cavity, and one (1) for empty cavity data. Data may be loaded using the **loadftpsi** command, or manually by pressing the Load Data button within an Acquisition Data window, or from the most recent measurement. This function is equivalent to clicking on the Analyze Peak button within the MST Custom Cavity application. Returns no value. (Refer to the *Custom Cavity Application Booklet*, OMP-0487, for additional information.)

asn*(RealExpr)*

Returns the arc-sine of the value of *RealExpr* in the current angle unit.

assign @FileVar to FileNameExpr [StrConst]

This statement first closes any open file already associated with *FileVar*. If *FileNameExpr* is of non-zero length, then the file with that pathname is opened and associated with *FileVar*. The optional *StrConst* specifies the mode:

Mode	Description
r	read (default)
w	write
a	append

atn*(RealExpr)*

Returns the arc-tangent of *RealExpr* parameter. The return v waitz waits for movement of the stage Z axis to complete. Returns no value. The value is in the range [-90, +90] degrees or [-2π, $+2\pi$] radians, depending on the current angle unit.

atn2*(RealExpr1, RealExpr2)*

Returns the arc-tangent of the value (Y/X) where Y is the value of *RealExpr1* and X is the value of RealExpr2. The signs of the two values are taken into account to determine quadrant information. The return value is in the range [-180, +180] degrees or [-2π, $+2\pi$] radians, depending on the current angle unit.

autoclampoff

Disables the motion controller's automatic clamping of air bearing stages.

autoclampon

Enables the motion controller's automatic clamping for air bearing stages.

autofocus

Causes MetroPro to perform an automatic focus adjustment. Equivalent to clicking on the MetroPro Focus button. Returns no value.

autofocusoff

Disables the motion controller's auto focus feature. The auto focus zero position needs to be set using setmcafzero for the motion controller's auto focus feature to be enabled.

autofocuson

Enables the motion controller's auto focus feature. The auto focus zero position needs to be set using **setmcafzero** for the motion controller's auto focus feature to be enabled.

autofocustilt

Causes MetroPro to perform an automatic focus and tilt adjustment. There is no MetroPro equivalent. Returns no value.

autolatcal(*IntExpr, RealExpr, StrExpr*)

This function calculates and sets the lateral resolution of the current objective selection. It is analogous to the 'Auto Cal' button currently available in the Microscope calibrator. Calibration is performed on the data obtained from measuring the Precision Lateral Calibration Standard. If *IntExpr* is zero (0), calibration is performed on data at the top level of MetroPro; otherwise, *IntExpr* must specify the ID of a data window obtained via the **getid** function (not **getwinid**). *RealExpr* specifies the mean spacing for the measured area of the standard. The mean spacing value can be read from the certificate of calibration included with each standard. *StrExpr* specifies the measurement units for the mean spacing of the measured area. The calculated resolution is returned in units specified by *StrExpr*.

autosavedata

Saves the current data using the values in the Auto Save Data Dir and Auto Save Data File controls. This resembles how data is saved by MetroPro when performing a measurement with the Auto Save Data control On. Note that the Auto Save Data File value is **not** incremented.

autotilt

Causes MetroPro to perform an automatic tilt adjustment. Equivalent to clicking on the MetroPro Auto Tilt button. Returns no value.

break

This command is only usable in the interactive mode. Refer to *Appendix B, Interactive Mode*, for information on using this command.

chr$(*IntExpr*)

Returns a string containing just one character whose ASCII numeric value is specified by the *IntExpr* parameter.

clampstage(*IntExpr*)

Causes the motion controller to clamp a single stage or all stages as indicated by *IntExpr*. The stage must have clamping enabled in the motor configuration file for this command to have any effect. For clamping a single stage, *IntExpr* has a valid range of [0, 7]. To clamp all stages, set *IntExpr* to -1.

clearmcafzero

Clears the motion controller's auto focus zero position. The auto focus zero position needs to be set using **setmcafzero** for the motion controller's auto focus feature to be enabled.

clearprocstats

Causes MetroPro to clear any process statistics including the stored results. Equivalent to clicking on the Clear button in a Process window. Returns no value.

clickon*(IntExpr)*

This function is used to "click on" a MetroPro button or selection control which makes it possible to perform some MetroPro operations for which there is no built-in MetroScript function. The *IntExpr* parameter specifies the ID of the button, or selection control obtained via the **getid** function. For example, to open the Lateral Calibrator window:

```
id = getid ("Buttons / Calibrate" )
clickon (id)
```

Note: This function cannot be used to close the application by "clicking on" the Close App button.

closeapp*(IntExpr)*

This function closes (iconizes) an open application. Returns no value.

The *IntExpr* parameter specifies the ID of the application, obtained via **getappid**. This function does nothing if the application is already closed. This function can only be used in a top-level MetroScript.

closewin*(IntExpr)*

This function closes (iconizes) an open window within an application. Returns no value.

The *IntExpr* parameter specifies the ID of the window obtained via **getwinid**. This function does nothing if the window is already closed.

contpattern

Continues the programmable stage pattern after it has been paused. Equivalent to clicking the MetroPro Cont Pattern button. Returns no value.

continue

This command is only usable in the interactive mode. Refer to *Appendix B*, *Interactive Mode*, for information on using this command.

convinten

Causes MetroPro to convert intensity data to height data. Equivalent to clicking the MetroPro Conv Intens to Data button. Returns no value.

cos*(RealExpr)*

Returns the cosine of the value of *RealExpr*. *RealExpr* must be in the current angle units.

cosh*(RealExpr)*

Returns the hyperbolic cosine of the value of *RealExpr.* *RealExpr* must be in the current angle units.

data *Const [, Const] ...*

This statement records data values for later use with the **read** statement. Any value may be prefixed with *IntConst* to make it be used *IntConst* times.

date$(RealExpr)

Returns a string containing the date corresponding to *RealExpr* number of seconds since the start of the Epoch. This function is commonly called with the argument value of the **timedate** function.

Examples:

? date$(0)

Thu Jan 1 1970

? date$(timedate)

Thu Feb 8 1996

deg

Sets degrees as the current angle unit for the trigonmetric functions and for stage axis positions. (See also the **rad** statement.)

del

This command is only usable in the interactive mode. Refer to *Appendix B*, *Interactive Mode*, for information on using this command.

delfid (*IntExpr*)

Deletes a fiducial. The ID of the fiducial is specified by *IntExpr*. See also **makefid**.

dialog*(StrExpr, IntExpr)*

Displays a MetroPro dialog box containing the specified *StrExpr* parameter and waits for user input. There are six modes selectable by the *IntExpr* parameter:

Mode	Description	Return Value
1	Normal confirm OK - green title bar	1
2	Error confirm OK - red title bar	1
3	Warning confirm OK - yellow title bar	1
4	Normal decide YES/NO - green title bar	1/ 0
5	Error decide YES/NO - red title bar	1/ 0
6	Warning decide YES/NO - yellow title bar	1/ 0

Clicking on the OK button or pressing any key, completes an OK dialog. Clicking on the YES or NO button completes a YES/NO dialog. Pressing the Enter key is equivalent to clicking on the YES button. For any dialog, pressing the ESC key produces the "Processing aborted" error.

dim DimItem [, DimItem] ...

This statement dimensions the specified *DimItems*. The same variable may be dimensioned multiple times during the execution of a program. A *DimItem* is one of the following:

StrVar dimensions a string variable for the default 32 characters.

*StrVar**[**IntConst**]*** dimensions a string variable for *IntConst* characters.

StrVar Subscripts dimensions an array of strings for the default 32 characters.

*StrVar Subscripts**[**IntConst**]*** dimensions an array of strings for *IntConst* characters.

RealVar Subscripts dimensions an array of real numbers.

Subscripts look like:

(*M*) specifies a one-dimensional array with subscripts 1 to *M*.
(*M, N*) specifies a two-dimensional array with subscripts 1 to *M* and 1 to *N*. *M* and *N* are normally *IntConsts*. They may also be *RealExprs*.

disable abort @*FileVar*

This statement disables handling of an abort request from a serial I/O device. *FileVar* must have been created by a previous **assign** statement.

edit

This command is only usable in interactive mode. Refer to *Appendix B, Interactive Mode*, for information on using this command.

editmasks*(IntExpr)*

Opens the MetroPro Mask Editor to allow the operator to edit masks or fiducials. *IntExpr* is a wait-for-close flag. If *IntExpr* is non-zero, the script pauses until the operator closes the Mask Editor.

editpatpos

Opens the MetroPro Pattern Position Editor window and allows user interaction. Completes when the user closes the window or presses the Escape key. Pressing the Esc key results in the "Processing aborted" error. Equivalent to clicking on the MetroPro Edit Pattern Pos button. Returns no value.

editpattern

Opens the MetroPro Pattern Editor window and allows user interaction. Completes when the user closes the window or presses the Escape key. Pressing the Esc key results in the "Processing aborted" error. Equivalent to clicking on the MetroPro Edit Pattern button. Returns no value.

else

Refer to the **if** command.

enable abort @*FileVar*

This statement enables handling of an abort request from a serial I/O device. *FileVar* must have been created by a previous **assign** statement.

end

This statement halts program execution. This is equivalent to the **stop** statement.

endif

Refer to the **if** command.

english

Sets inches as the current linear unit for stage axis positions. (*See also:* the **metric** statement.)

enter @*FileVar* ; *Var* [, *Var*] ...

This statement reads data from the file associated with *FileVar* into the specified *Vars*. This is similar to the **input** statement.

enterline @FileVar ; StrVar

Returns a single line from the file associated with *FileVar* into the string variable *StrVar*.

errl*(LineId)*

Returns 1 if the most recent error occurred in the specified line, otherwise returns 0. Using this function with a *LineLabel* instead of a *LineNum* avoids problems caused by renumbering a program.

errln

Returns the line number of the most recent error.

errm$

Returns the most recent error message from a running program.

errn

Returns the number of the most recent error from a running program. The possible values are:

Value	Error
101	Attempt to divide by zero.
102	Ran out of input during read.
103	Invalid exponentiation.
104	Floating-point overflow.
105	Argument out of range.
106	File access error.
107	Invalid data during read.
108	Subscript out of range.
1004	Various fatal (unrecoverable) program errors.
1005	Various fatal (unrecoverable) internal errors.

error return

See **on error gosub**.

execute*(StrExpr)*

The string parameter is passed to the DOS shell and executed as a command. Returns the integer exit status of the command. Usually a zero exit status means successful execution.

exit

This command is only usable in interactive mode. Refer to *Appendix B***,** *Interactive Mode*, for information on using this command.

exitmp*(IntExpr)*

Exits immediately from MetroPro to the operating system with a status value specified by *IntExpr*. The useful range of *IntExpr* is 0 to 127.

exp*(RealExpr)*

Returns the natural exponentiation of the value of *RealExpr.*

findfid*(IntExpr, RealExpr1, RealExpr2, RealExpr3, RealExpr4, StrExpr)*

This function searches for a fiducial in a MetroPro data matrix (map). There is no MetroPro equivalent to this function. Returns no value.

The search is controlled by parameters previously established by the functions: **setfidsize, setfidthresh**, and **setfidpat**. The results of the search are subsequently obtained via the functions: **getfidctrx**, **getfidctry**, **getfidorgx**, **getfidorgy**, **getfidsizx**, and **getfidsizy**.

The data matrix is specified by the *IntExpr* parameter as follows: If *IntExpr* is zero (0), the matrix is the height map at the top level of the MetroPro app (unprocessed phase data) obtained by measuring, loading, or generating. Otherwise, *IntExpr* must specify the ID of a data window, obtained via the function **getid** (not **getwinid**), and the matrix is the one associated with that data window.

The remaining parameters specify the location and dimensions of a rectangular region within the matrix for the purpose of limiting the search.

The X and Y coordinates of the center of the rectangle are specified by *RealExpr1* and *RealExpr2*, respectively. The rectangle X and Y dimensions are specified by *RealExpr3* and *RealExpr4*, respectively. The measurement unit is specified by *StrExpr*. If null, the default unit is pixels.

Either of the dimension parameters may be set to zero (0) to obtain default values for both the dimension and the center coordinate. The default center coordinates are the same as the values returned by functions **getdatactrx** and **getdatactry**. The default X and Y dimensions are the same as the values returned by the functions **getdatasizx** and **getdatasizy**.

The coordinate system is consistent with the Mask Editor display, which corresponds to the image from an actual or virtual camera. The location of the origin (0,0) of the coordinate system depends on the measurement unit.

If the unit is pixels, the origin of the coordinate system is the lower left corner of the image. X and Y are more positive to the right and up, respectively.

If the unit is other than pixels, the origin of the coordinate system is the center of the image. X and Y are more positive to the right and up, and more negative to the left and down, respectively.

The dimensions of the coordinate system are the effective dimensions of an image from the camera used to measure the data (or the virtual camera used to generate the data).

flipdatax

Causes MetroPro to flip the current data in the X dimension. Equivalent to clicking on the MetroPro Flip Data X button. Returns no value.

flipdatay

Causes MetroPro to flip the current data in the Y dimension. Equivalent to clicking on the MetroPro Flip Data Y button. Returns no value.

for RealVar = RealExpr1 to RealExpr2 [step RealExpr3]

This statement repeats a group of instructions a specified number of times. There should always be an associated **next** statement.

Let *Step* and *Limit* be hidden variables associated with this statement and the corresponding **next** statement. If **step** *RealExpr3* is specified, *Step* is assigned the value of *RealExpr3*, otherwise 1. *RealVar* is assigned the value of *RealExpr1* - *Step*. *Limit* is assigned the value of *RealExpr2*. Execution skips to the **next** statement specifying the same *RealVar*. This statement may also be used with an *IntVar* and *IntExprs*.

generate

Causes MetroPro to generate data. Equivalent to clicking on the MetroPro Generate button. Returns no value.

gensyschar*(FileNameExpr)*
gensyschar*(FileNameExpr, SwliFileNameExpr)*

This command generates a system characterization file in the current working directory as specified by *FileNameExpr.* The file is created either by performing a measurement of an SiC Reference Flat, or by loading a SWLI data file. If using a SWLI file for generating the system characterization file, be sure to use ".syc" as the file extension. Set the following controls as specified below:

FDA Res: Films
DiscrMode: RAM
Films Mode: Top Surface

In addition, use the objective that will be used when measuring the part. The Scan Direction control must be set the same as well. If *FileNameExpr* is set to "@," the actual filename will be automatically generated based on the objective magnification and scan direction. If the optional parameter *SwliFileNameExpr* is specified, the system characterization file is generated using the SWLI data *SwliFileNameExpr*. If it is not specified, a measurement is taken to generate the file. No value is returned.

getannotid*(IntExpr, StrExpr)*

This function returns the ID of a named annotation box within a window. The ID can then be used in calls to the **setstr** or **getval$** functions. The *IntExpr* parameter specifies the ID of the window, obtained via **getwinid**. The *StrExpr* parameter specifies the name of the annotation box.

Note that blanks and slash "/" characters are not significant in *StrExpr*. An error is generated if there is no annotation box with the specified name. If there are multiple annotation boxes with the specified name, this function will return the ID of the first one found.

getappid(*StrExpr*)

This function returns the ID of a loaded application. The ID can then be used in calls to various application control functions. The *StrExpr* parameter specifies the file name used to load the application. An error is generated if there is no loaded application with the specified file name. This function can only be used in a top-level MetroScript.

getavmode

Returns the current align/view mode of the GPI. It returns an integer value of zero (0) for view mode, and one (1) for align mode.

getbindir$

Returns the pathname of the directory containing executable programs (binaries) for the current MetroPro installation.

getcamres(*StrExpr*)

Returns the camera resolution in the unit specified by the *StrExpr* parameter. If there is an active instrument, the returned value is the effective spacing of camera pixels based on the Camera Mode control, the Lateral Calibrator, and if the instrument is a microscope, the objective selection and the Image Zoom control. Otherwise, if there is a data set measured or loaded, the returned value is the effective lateral spacing of points in the data. Otherwise, the Camera Res control value is returned (only available in applications that can generate data). Returns zero (0) if unknown.

getcamsizx(*StrExpr*)
getcamsizy(*StrExpr*)

These functions return the camera dimensions in the unit specified by the *StrExpr* parameter. If there is an active instrument, the returned values are the effective dimensions of an image from the camera based on the Camera Mode control, the Lateral Calibrator, and if the instrument is a microscope, the objective selection and the Image Zoom control. Otherwise, if there is a data set measured or loaded, the returned values are the effective dimensions of an image from the camera used to measure the data (or the virtual camera used to generate the data). Or the returned values may be based on the Camera Size X, Camera Size Y, and Camera Res control values when used in applications that can generate data.

getdatactrx(*IntExpr, StrExpr*)
getdatactry(*IntExpr*, *StrExpr*)
getdataorgx(*IntExpr*, *StrExpr*)
getdataorgy(*IntExpr*, *StrExpr*)

getdatasizx(*IntExpr*, *StrExpr*)
getdatasizy(*IntExpr*, *StrExpr*)

These functions return the position and dimensions of a MetroPro data matrix (map). The data matrix is specified by the *IntExpr* parameter as follows: If *IntExpr* is zero (0), the matrix is the height map at the top level of the MetroPro app obtained by measuring, loading, or generating. Otherwise, *IntExpr* must specify the ID of a data window, obtained via the function **getid** (not **getwinid**), and the matrix is the one associated with that data window.

The measurement unit is specified by the *StrExpr* parameter. If null, the default unit is pixels.

The values returned from **getdatactrx** and **getdatactry** are the coordinates of the geometric center of the data matrix.

The values returned from **getdataorgx** and **getdataorgy** are the coordinates of the lower left corner of the data matrix.

The values returned from **getdatasizx** and **getdatasizy** are the dimensions of the data matrix.

The coordinate system is consistent with the Mask Editor display, which corresponds to the image from an actual or virtual camera. The location of the origin (0,0) of the coordinate system depends on the measurement unit.

If the unit is pixels, the origin of the coordinate system is the lower left corner of the image. X and Y are more positive to the right and up, respectively. If the unit is other than pixels, the origin of the coordinate system is the center of the image. X and Y are more positive to the right and up, and more negative to the left and down, respectively.

The dimensions of the coordinate system are as described for the **getcamsizx** and **getcamsizy** functions.

getdir$(*StrExpr*)

This command returns the directory path used by MetroPro to save or load files of a specified type. The file type is specified by *StrExpr* and must be one of the names in the following table. The case of letters and embedded blanks is not significant. For example, “Q-DAS Report” and “q-dasreport” are the same. Use **setdir** to change the directory paths. See also **getbindir$** and **getwrkdir$**.

App	Model	Report
Bitmap	Pattern	Script
Config	Pattern Position	Settings
Data	PSF	SWLI Data
Fiducials	Q-DAS Report	TIFF
Masks	Regions	Zernikes

getdiskfree(*StrExpr*)

Returns the number of bytes free on the specified disk. The *StrExpr* specifies the disk drive letter. If *StrExpr* is a null string, the “current” drive is assumed.

getdistsys

Returns the Dist Sys calibration value for a Simetra FTP instrument in inches. This value is normally specified in the system calibration file.

getdmipos(*StrExpr*)

This function returns the current position reported by a distance measuring instrument (DMI). The measurement unit is specified by the *StrExpr* parameter, which cannot be null.

getdmiscale

Returns the DMI scale factor calibration value for a Simetra FTP instrument. This value is normally specified in the system calibration file.

getdmitype$

Returns a string indicating the distance measuring instrument (DMI), if any.

getfidctrx(*StrExpr*)
getfidctry(*StrExpr*)
getfidorgx(*StrExpr*)
getfidorgy(*StrExpr*)
getfidsizx(*StrExpr*)
getfidsizy(*StrExpr*)

These functions return the location and size of a fiducial found by the function **findfid**. The measurement unit is specified by the *StrExpr* parameter. If null, the default unit is pixels. The values returned from **getfidctrx** and **getfidctry** are the coordinates of the centroid of the fiducial.

The values returned from **getfidorgx** and **getfidorgy** are the coordinates of the lower left corner of the fiducial region. The values returned from **getfidsizx** and **getfidsizy** are the dimensions of the fiducial region. If the fiducial was found using **setfidthresh** Mode 2, the fiducial region is defined by the size of the pattern matrix specified by **setfidpat**. Otherwise the fiducial region is defined as the minimal enclosing rectangle for the points found.

The coordinate system is consistent with the Mask Editor display that corresponds to the image from an actual or virtual camera. The location of the origin (0,0) of the coordinate system depends on the measurement unit.

If the unit is pixels, the origin of the coordinate system is the lower left corner of the image. X and Y are more positive to the right and up, respectively. If the unit is other than pixels, the origin of the coordinate system is the center of the image. X and Y are more positive to the right and up, and more negative to the left and down, respectively.

The dimensions of the coordinate system are as described for the **getcamsizx** and **getcamsizy** functions.

getftpsipeakbynum*(Intexpr1, RealExpr1, IntExpr2, IntExpr3, RealExpr2, RealExpr3, StrExpr)*

Returns the location of a peak based upon an OPD spectrum calculation of a pixel at the location of the Test Calibration marker.

IntExpr1 (Data Source)

Specifies the source of the MST data. For all MST applications besides MST Homogeneity, the source must be zero. When loading data into the MST Homogeneity application, set *IntExpr* to zero for full cavity, and 1 for the empty cavity data.

RealExpr1 (Peak Threshold)

Threshold for peak detection. Possible values are between zero (0) and one (1). Indicates the required signal strength to be classified as a peak. The strength is relative to the highest peak.

IntExpr2 (Peak Number)

Specifies which peak out of the number of peaks specified by *IntExpr3* is found. The location of the selected peak is returned by this command.

IntExpr3 (Number of Peaks)

Specifies the number of expected peaks. Peaks are found based upon the highest signal strength first.

RealExpr2 (Lower Search Range)

Specifies the Optical Path Length that is the lower boundary when searching for peaks.

RealExpr3 (Upper Search Range)

Specifies the Optical Path Length that is the upper boundary when searching for peaks.

StrExpr (Units Specifier)

Specifies the unit type for all input parameters and returned values.

The following example illustrates how to use the **getftpsipeakbynum** command to return the OPD of a two surface gap. The example is based upon a 4-surface measurement as outlined in the *Test Calibration Pixel Trace Window* section of OMP-0487.

The command as shown below returns the optical path distance for peak 4 of 6, found between 0 and 400 mm. In the 4-surface example, peak number 4 corresponds to gap 3-4 (back surface of test part).

```
dataSRC      = 0
peakThresh   = 0.15
peakNum      = 4
numPeaks     = 6
lowerOPD     = 0
upperOPD     = 400
unitStr$     = “mm”
peakOPD = getftpsipeakbynum( dataSrc, peakThresh, peakNum, numPeaks, minOPD, maxOPD,
unitStr$ )
```

The value assigned to peak OPD would be 195.9 mm given the example data shown in OMP-0487.

This function is available in the following MST applications: Two Surface, Parallel Surface, Homogeneity and Custom Cavity.

getftpsipeakbypos*(IntExpr, RealExpr1, RealExpr2, StrExpr)*

Returns the location of a peak based upon an OPD spectrum calculation of a pixel at the location of the Test Calibration marker.

IntExpr1 (Data Source)

Specifies the source of the MST data. For all MST applications besides Homogeneity, the source must be zero (0). When loading data into the Homogeneity application, set *IntExpr* to zero for full cavity and one (1) for empty cavity data.

RealExpr1 (Lower Search Range)

Specifies the Optical Path Length that is the lower boundary when searching for peaks.

RealExpr 2 (Upper Search Range)

Specifies the Optical Path Length that is the upper boundary when searching for peaks.

StrExpr (Units Specifier)
Specifies the units type for all input parameters and the returned value.

The following example illustrates how to use the **getftpsipeakbypos** command to get the OPD of a 2-surface gap. The example is based upon a 4-surface measurement outlined in the *Test Calibration Pixel Trace Window* section of
OMP-0487.

The command, as shown below, returns the optical path distance for the strongest peak found between 150 and 250 mm. In this 4-surface example, the strongest peak in the search range corresponds to gap 3-4 (back surface of the test part).

```
dataSrc     = 0
lowerOPD    = 150
upperOPD    = 250
unitStr$    = "mm"
peakOPD = getftpsipeakbypos( dataSrc, lowerOPD, upperOPD, unitStr$ )
```

This function is available in the following MST applications: Two Surface, Parallel Surface, Homogeneity, and Custom Cavity.

getftpsipeakpower*(IntExpr, RealExpr, StrExpr)*

Returns the power of an OPD transform calculation at the distance given by *RealExpr*. The calculation occurs for a pixel at the location of the Test Calibration marker. *StrExpr* specifies the units.

There must be data available to perform the analysis. *IntExpr* specifies the source of the MST data. For all MST applications besides Homogeneity, the source must be zero (0). When loading data into the Homogeneity application, set *IntExpr* to zero for full cavity and one (1) for empty cavity data. Data may be loaded using the **loadftpsi** command; manually loaded by clicking the Load Data button within an Acquisition Data window, or from the most recent measurement. There is no equivalent function in MetroPro.

This function is available in the following MST applications: Two Surface, Parallel Surface, Homogeneity, and Custom Cavity.

getgpizoom (*IntExpr*)

Returns the current position of a GPI encoded zoom lens.

If *IntExpr* is zero, the position is in motor counts. Otherwise, if IntExpr is non-zero, the position value is a magnification. An error is generated if magnification calibration data does not exist.

(This command applies to GPI or MST instruments with an encoded zoom lens.)

getgpizoommax (*IntExpr)*

Returns the maximum position of a GPI encoded zoom lens.

If IntExpr is zero, the position is in motor counts. Otherwise, if the IntExpr is non-zero, the position value is a magnification. An error is generated if magnification calibration data does not exist.

(This command applies to GPI or MST instruments with an encoded zoom lens.)

getgpizoommin (*IntExpr*)

Returns the minimum position of a GPI encoded zoom lens.

If IntExpr is zero, the position is in motor counts. Otherwise, if IntExpr is non-zero, the position value is a magnification. An error is generated if magnification calibration data does not exist.

(This command applies to GPI or MST instruments with an encoded zoom lens.)

getid*(StrExpr)*
getid*(IntExpr, StrExpr)*

Returns an integer that is the ID for a MetroPro window, control, attribute, result, button, or limits object. An ID value is used when calling functions **getval**, **getval$**, **savedata**, **setnum**, or **setstr**.

In the first form *StrExpr* is a pathname like those found in MetroPro setting files except that the forward slash "/" is used as a delimiter instead of the backward slash "\".

Examples:
```
id = getid( "Controls / Miscellaneous / Lot Num" )
setstr( id, "Lot 29" )
id = getid( "Surface Map / Controls / Trim" )
setnum( id, 2, "")
```

The second form of **getid** provides access to controls and attributes that appear in the Report, Q-DAS Report and Video Monitor windows. In this form, the *IntExpr* specifies a window ID obtained via the **getwinid** function and *StrExpr* specifies a pathname. Q-DAS Report controls can be accessed by starting the path with "Parts Controls," "Results Controls," or "Data Attr Controls."

Example:
```
wid = getwinid( "My Report" )
id = getid( wid, "Controls / Log File" )
id = getid( wid, "Parts Controls / Test Part Number" )
```

getlight

Returns the current instrument light level as a percentage.

getmaskctrx(*IntExpr*, *StrExpr*)
getmaskctry(*IntExpr*, *StrExpr*)
getmasksizx(*IntExpr*, *StrExpr*)
getmasksizy(*IntExpr*, *StrExpr*)

These functions return the locations and sizes of a mask figure. The *IntExpr* parameter specifies the ID of a mask figure obtained via **getmaskid**. The measurement unit is specified by the *StrExpr* parameter. If null, the default unit is pixels.

The values returned from **getmaskctrx** and **getmaskctry** are the coordinates of the center of the figure.

The values returned from **getmasksizx** and **getmasksizy** are the dimensions of the minimal enclosing rectangle for the figure.

The coordinate system is consistent with the Mask Editor display that corresponds to the image from an actual or virtual camera. The location of the origin (0,0) of the coordinate system depends on the measurement unit.

If the unit is pixels, the origin of the coordinate system is the lower left corner of the image. X and Y are more positive to the right and up, respectively. If the unit is other than pixels, the origin of the coordinate system is the center of the image. X and Y are more positive to the right and up, and more negative to the left and down, respectively.

The dimensions of the coordinate system are as described for the **getcamsizx** and **getcamsizy** functions.

getmaskid*(RealExpr1, RealExpr2, StrExpr)*

Returns the ID for the mask figure that is closest to specified coordinates. Returns an integer. An ID value is required when calling the various functions for querying and manipulating mask figures.

The X and Y coordinates are specified by the *RealExpr1* and *RealExpr2* parameters, respectively. The measurement unit is specified by the *StrExpr* parameter. If null, the default unit is pixels. The coordinate values are for a point close to the center of the target figure.

The coordinate system is consistent with the Mask Editor display, which corresponds to the image from an actual or virtual camera. The location of the origin (0,0) of the coordinate system depends on the measurement unit.

If the unit is pixels, the origin (0,0) of the coordinate system is the lower left corner of the image. X and Y are more positive to the right and up, respectively. If the unit is other than pixels, the origin (0,0) of the coordinate system is the center of the image. X and Y are more positive to the right and up, and more negative to the left and down, respectively.

The dimensions of the coordinate system are as described for the **getcamsizx** and **getcamsizy** functions.

getmicrolynxpos

Returns the current position of the Micro LYNX stage in the current linear unit, either inches or millimeters. This operation is equivalent to clicking the Read button in the MetroPro Micro LYNX window. The stage must be homed first.

getnum*(IntExpr, StrExpr)*

See **getval**.

getosname$

Returns the name of the host operating system.

getpatval*(StrExpr1, Var, StrExpr2)*

This function gets a value (or an array of values) from the current stage control pattern. (This is the pattern viewable in the Pattern Editor.) *StrExpr1* specifies the value name. (See the table of pattern values following the **getsubpatval** command.) *Var* specifies a MetroScript variable to which the value or values are assigned. *StrExpr2* specifies the unit, if relevant. Returns no value.

getpos*(IntExpr1, IntExpr2)*
getpos*(StrExpr, IntExpr2)*

Returns the current position of a stage axis.

The axis may be specified as a letter (a one-character string) using *StrExpr,* or as an axis number using *IntExpr1*. Axis numbers are normally in the range [0, 7]. Axis letters are normally "X," "Y," "P," "R," "T," or "r."

The *IntExpr2* parameter specifies the unit. If the value of *IntExpr2* is negative one (–1), the position is returned in stepping motor controller counts. Otherwise the position is returned in the unit appropriate for the axis type.

getppos

Returns the current position of the Pitch Axis Stage in the current angle unit.

getramsize

Returns the amount of physical memory in bytes.

getrpos

Returns the current position of the Roll Stage Axis in the current angle unit.

getrsmode

Returns the current ring/spot mode for the VeriFire/GPI. This command returns an integer value of zero (0) for Ring mode and one (1) for Spot mode.

getstr$(*IntExpr*)

See **getval$**.

gettpos

Returns the current position of the Theta stage axis in the current angle unit.

getturpos(*IntExpr1*)

Returns the current position of the motorized turret on a microscope. The NewView 6000 return value range is one through six (1—6). The NewView 5000 return value range is one through five (1—5).

getscancal

Returns the scan calibration factor for a scanning type instrument. This value is normally specified in the system calibration file.

getsubpatval*(StrExpr1, Var, StrExpr2)*

This function gets a value (or an array of values) from the current stage control subpattern. (This is the pattern in the hidden subpattern holding area.) *StrExpr1* specifies the value name. (See the table of values below.) *Var* specifies a MetroScript variable to which the value or values are assigned. *StrExpr2* specifies the unit, if relevant. Returns no value. (Refer to *Appendix D* for additional information on subpattern functions.)

Name	Get / Set	Get Var Type	Unit

Pattern Values Table

AlignArrayX	Get Only	1D Real Array	Linear
AlignArrayY	Get Only	1D Real Array	Linear
ColSpace	Get / Set	Real	Linear
Dirname	Get Only	Str	
EndCommand	Get / Set	Str	
FailureAction	Get / Set	Real	
Filename	Get Only	Str	
LoadX	Get / Set	Real	Linear
LoadY	Get / Set	Real	Linear
NumAlign	Get Only	Real	
NumCircles	Get / Set	Real	
NumCols	Get / Set	Real	
NumPos	Get Only	Real	
NumPosIncl	Get Only	Real	
NumRadials	Get / Set	Real	
NumRows	Get / Set	Real	
Operation	Get / Set	Real	
Order	Get / Set	Real	
OriginX	Get / Set	Real	Linear
OriginY	Get / Set	Real	Linear
PatType	Get / Set	Real	
PauseAfter	Get / Set	Real	
PauseBefore	Get / Set	Real	
PosArrayId	Get Only	1D Real Array	
PosArrayX	Get Only	1D Real Array	Linear
PosArrayY	Get Only	1D Real Array	Linear
Prompt	Get / Set	Real	
RectOriginLoc	Get / Set	Real	
Rotation	Get / Set	Real	Angle
RowSpace	Get / Set	Real	Linear
ScriptFilename	Get / Set	Str	
SecColSpace	Get / Set	Real	Linear
SecRowSpace	Get / Set	Real	Linear
StartCommand	Get / Set	Str	

getsyserrfn$

Returns the name of the system error data file used for a current or subsequent measurement if the Subtract Sys Err control is On. If the first letter of the Sys Err File control setting is "@," the return value is an automatically generated file name. Otherwise, the return value is the same as the control setting.

getval(*IntExpr, StrExpr*)
getnum(*IntExpr, StrExpr*)

The two forms of this command are equivalent. Returns the numeric value of the MetroPro control, attribute, or result referenced by ID *IntExpr* in the unit specified by *StrExpr*. This function may be used to obtain the value of numeric, selection, and Boolean-type items. Use **getstr$** or **getval$** to obtain the value of string type items and limits. The return value depends on the item type as follows:

Item Type	Return Value
Numeric	Value in the unit specified by *StrExpr*.
Selection	Index of the selection (1..N). *StrExpr* is ignored.
Boolean	1 for "Yes," 0 for "No" *StrExpr* is ignored.

Certain numeric controls can be disabled by setting them to blank. If a numeric control is disabled, this function returns **maxreal**.

getval$(*IntExpr*)
getstr$(*IntExpr*)

The two forms of this command are equivalent. Returns the string value of the MetroPro control, attribute, result, or limits referenced by ID *IntExpr*. This function may be used to obtain the value of string, selection, and Boolean type items as well as limits. Use **getnum** or **getval** to obtain the value of numeric type items. The return value depends on the item type as follows:

Item Type	Return Value
String	The displayed string.
Selection	The displayed selection name.
Boolean	"Yes" or "No".
Limits	Limits description as in a MetroPro settings file. Example: "[1.23, 4.56] uin"

getwinid(*StrExpr*)

Returns an integer that is the ID of a window within a MetroPro application. The ID can then be used in calls to **getid** and the various window control functions. The *StrExpr* parameter specifies the title of the window. Note that blanks and slash "/" characters are not significant in *StrExpr*. An error is generated if there is no window with the specified title, or if there are multiple windows with the specified title.

Example: wid = getwinid(“Surface Map”)
openwind(wid)

getwrkdir$

Returns the name of the MetroPro working directory.

getxpos

Returns the current position of the X Stage Axis in the current linear unit.

getypos

Returns the current position of the Y Stage Axis in the current linear unit.

getzernikes

This function is equivalent to clicking on the Get Zernikes button in MetroPro.

getzoom

Returns the current image zoom value as a real number.

getzpos

Returns the current position of the Z Stage Axis in the current linear unit.

gosub ***LineId***

This statement executes a subroutine beginning at the specified line. An error occurs if the specified line does not exist. The location of the statement following the **gosub** is saved, and when a **return** statement is executed, execution proceeds from the saved location. (See also, **loadsub** in *Appendix B.*)

goto ***LineId***

This statement executes the statements beginning at the specified line. An error occurs if the specified line does not exist.

gotopatorg

Moves the programmable stage X and Y axes to the pattern origin position. Equivalent to clicking the MetroPro Goto Origin button.

gotopatpos*(IntExpr)*

Moves the stage X and Y axes to a pattern position. The *IntExpr* parameter specifies the pattern position as an index into the array of included pattern positions. A value of 1 specifies the first position that is visited when the pattern is run.

homeaxis*(StrExpr)*
homeaxis*(IntExpr)*

These functions cause a programmable stage axis to establish a "home" position. The axis may be specified as a label using *StrExpr*, or as an axis number using *IntExpr*. Axis labels are normally 'X,' 'Y,' 'Z' etc. Axis numbers are normally in the range [0,7]. The operation may or may not cause stage movement, depending on how the axis is configured. These functions return no value.

homemicrolynx

Causes the Micro LYNX stage to establish the "home" position in the Z axis. This operation is equivalent to clicking on the MetroPro Home button in the Micro LYNX window.

homestage

Causes the programmable stage to establish the “home” positions of the X and Y axes. Each of the X and Y axes will move to one extreme of travel. This operation is equivalent to clicking on the MetroPro Home Stage button.

homezoom

On systems with the motorized zoom option, this command cycles through the available image zoom lenses and updates the image zoom list. This command is ignored if the zoom is not motorized.

if *LogicExpr* **then** *Stmt1 [* **else** *Stmt2]*

This statement is the single-line form of the "if-then" statement. The *LogicExpr* is evaluated. If *LogicExpr* is true, *Stmt1* is executed. Otherwise, *Stmt2* is executed, if it is specified. The block form of this statement is described below.

if *LogicExpr* **then**
[**else** *]*
endif

This statement is the block form of the "if-then" statement. The *LogicExpr* is evaluated. If *LogicExpr* is true, subsequent statements are executed up to the corresponding **else** statement or **endif** statement if the **else** statement is omitted. Otherwise, execution continues with statements after the corresponding **else** statement, after the corresponding **endif** statement if the **else** statement is omitted. See also the single-line version of this statement described above.

input *StrConst* , *Var [* , *Var]* ...

This statement is the same as the **input** statement except that the user is prompted with *StrConst* followed by a question mark.

input *Var [* , *Var]*

This statement prompts the user with a question mark and the values of the specified *Vars* are read from the terminal. If a string value is expected, quotes are not required, and the string may contain C language escape codes. For input from files, use the **enter** statement.

int(*RealExpr*)

Returns the integer part of the *RealExpr* parameter.

invertdata

Causes MetroPro to invert the current data. Equivalent to clicking on the MetroPro Invert Data button. Returns no value.

isaxisactive(*StrExpr*)
isaxisactive(*IntExpr*)

This command checks whether or not the specified stage axis is active. The axis may be specified as a label using *StrExpr*, or as an axis number using *IntExpr*. Axis labels are: X, Y, Z, P, R, r, T, A, and B. Axis numbers range from 0 to 7. The return value is zero (0) if the axis is inactive, and one (1) if the axis is active.

ismcafenabled

This command checks whether or not the motion controller's auto focus feature is enabled. The return value is zero (0) if disabled, and one (1) if enabled.

ismcafzeroset

This command checks whether or not the motion controller's auto focus zero position is set. The return value is zero (0) if the zero position is not set, and one (1) if the zero position is set.

israsactive

This command checks whether or not the script is being run through remote access. The return value is zero (0) if remote access is not running, or if the server is suspended, and one (1) if remote access is running.

isstageclamped(*StrExpr*)
isstageclamped(*IntExpr*)

This command checks whether or not the specified stage axis is clamped. The axis may be specified as a label using *StrExpr,* or as an axis number using *IntExpr*. Axis labels are: X, Y, Z, P, R, r, T, A, and B. Axis numbers range from zero (0) to 7. The return value is zero (0) if the axis is not clamped, and one (1) if the axis is clamped.

joyoff

Disables the Motion Controller joysticks.

joyon

Enables the Motion Controller joysticks. If this function is called while stage motion is in progress, an error is generated.

len(*StrExpr*)

Returns the number of characters in the *StrExpr* parameter.

lgt(*RealExpr*)

Returns the base 10 logarithm of the *RealExpr* parameter.

llcdialog

Opens the MetroPro Light Level for Measure window and allows user interaction. Completes when the window is closed. It is equivalent to pressing the F4 function key. This command should only be used within a script. Using **llcdialog** in the interactive mode of MetroScript as a single instruction may yield unexpected results. Returns no value.

load

This command is only usable in the interactive mode. Refer to *Appendix B*, *Interactive Mode*, for information on using this command.

loadapp(*FileNameExpr*)

Causes MetroPro to load the application file specified by the *FileNameExpr* parameter. Equivalent to selecting the Load Application item in the desktop pop-up menu and then selecting *FileNameExpr* as the file name in the dialog box. Returns no value.

loadbitmap*(IntExpr, StrExpr)*

This command loads and displays a bitmap file. *IntExpr* must be the ID of a MetroPro object that supports display of a bitmap, such as a bitmap window, button, or an icon. The ID may be obtained using either the getid or getwinid functions. *StrExpr* specifies the the bitmap filename. The file must be located in MetroPro's current working directory. A bitmap cannot be resized from within MetroScript. This must be done with an external image editing program prior to loading.

loaddata(*FileNameExpr*)

Causes MetroPro to load the data file specified by the *FileNameExpr* parameter. Equivalent to clicking on the Load Data button and then selecting *FileNameExpr* as the file name in the dialog box. Returns no value.

loaddata(*IntExpr, FileNameExpr*)

With this form of the command, the value of the IntExpr parameter is the ID of a data cell returned by the getid function. It provides a means to load data from a file into a specific data window. The only allowed target data windows are those that provide a Load Data button. This form of the command is primarily intended for future use since the only supported target data window currently available is the Rotationally Variant Surface map, which is part of the Asphere measurement applications.

loadftpsi(*IntExpr, FileNameExpr*)

Causes MetroPro to load the MST data file specified by the *FileNameExpr* parameter.

IntExpr specifies the destination of the MST data. For all MST applications besides Homogeneity, the destination must be zero (0). When loading data into the Homogeneity application, set *IntExpr* to zero for full cavity, and one (1) for empty cavity data. This function is equivalent to clicking on the LoadAcqData button within a Data Acquisition window and then selecting *FileName Expr* as the file name in the dialog box. Returns no value. An MST data file contains the raw data needed to calculate height maps. MST data is normally only saved or loaded for diagnostic purposes.

This function is available in the following MST applications: Two Surface, Parallel Surface, Homogeneity, and Custom Cavity.

loadmasks(*FileNameExpr*)

Causes the MetroPro Mask Editor to load the mask file specified by the *FileNameExpr* parameter. Equivalent to clicking on the Load button and then selecting *FileNameExpr* as the file name in the dialog box. Returns no value.

loadmstcalmarkers(*FileNameExpr*)

This command loads the calibration marker file (*.clm) specified by the *FileNameExpr* parameter. It is equivalent to clicking the Load button within the Mask Editor while in Cal Markers mode and then selecting *FileNameExpr* as the file name in the dialog box. This function does not return a value.

This function is available in all MST applications.

loadpartconfig(*FileNameExpr*)

Loads into MetroPro the settings file specified by *FileNameExp*r. Equivalent to clicking on the MetroPro Load Part Config button and then selecting *FileNameExpr* as the file name in the dialog box. Returns no value. For most application types, this is equivalent to the **loadsettings** function. For the FTP application only, this function performs additional operations, including:

- The state of the Mask Editor is established as follows. If an Auto Load Masks file is specified, a mask file is loaded. Otherwise all masks are cleared.
- The FTP instrument left and right axes are adjusted to match the Setup Type control and the relevant Measure Pos controls.

loadpatpos(*FileNameExpr*)

Causes the MetroPro Pattern Position Editor to load the pattern position file specified by the *FileNameExpr* parameter. Equivalent to selecting the Load item in the Position Editor pop-up menu and then selecting *FileNameExpr* as the file name in the dialog box. Returns no value.

loadpatstatus(*FileNameExpr*)

Causes the MetroPro Pattern Position Status window to load the pattern position status file specified by the *FileNameExpr* parameter. There is no equivalent operation in MetroPro. Returns no value.

loadpattern(*FileNameExpr*)

Causes the MetroPro Pattern Editor to load the pattern file specified by the *FileNameExpr* parameter. Equivalent to clicking on the Load button in the Pattern Editor and then selecting *FileNameExpr* as the file name in the dialog box. Returns no value.

loadsettings(*FileNameExpr*)

Loads the settings file specified by *FileNameExpr* into MetroPro. Equivalent to clicking on the MetroPro Load Settings button and then selecting *FileNameExpr* as the file name in the dialog box. It also updates the Settings File attribute in MetroPro. Returns no value.

loadsub

Refer to *Appendix B, Interactive Mode*, for information on using this command.

loadsubpat(*FileNameExpr*)

Causes the pattern file specified by the *FileNameExpr* parameter to be loaded into the hidden subpattern holding area. There is no equivalent in MetroPro. Returns no value. (Refer to *Appendix D* for additional information on subpattern functions.)

loadswli(*FileNameExpr*)

Causes MetroPro to load the SWLI (scanning white light interferometry) data file specified by the *FileNameExpr* parameter and then perform FDA (frequency domain analysis) to obtain a height map. This function is equivalent to clicking on the Load SWLI button and then selecting *FileNameExpr* as the file name in the dialog box. Returns no value. A SWLI data file contains the raw data needed by FDA to calculate a height map. SWLI data is normally only saved or loaded for diagnostic purposes.

lockgpizoom

Locks the position of a GPI encoded zoom lens so that it cannot be inadvertently changed. This also prevents any variation of the effective magnification value due to encoder noise.

(This command applies to GPI or MST instruments with an encoded zoom lens.)

log(*RealExpr*)

Returns the natural logarithm of the *RealExpr* parameter.

log2(*RealExpr*)

Returns the base 2 logarithm of the *RealExpr* parameter.

logfile is *FileNameExpr*

This statement duplicates terminal output to a new file specified by *FileNameExpr*.

If *FileNameExpr* is a zero length string, any previous **logfile is** file is closed. This statement allows the "$" character within the file/path name.

logreports

Causes MetroPro to log reports in all Report windows in which the Logging control is turned on.

lwc$(*StrExpr*)

Returns a copy of the *StrExpr* parameter with all upper case letters converted to lower case.

makefid(*RealExpr1*, *RealExpr2*, *StrExpr*, *IntExpr1*, *IntExpr2*)

Makes a new fiducial. The center (X,Y) coordinates are specified by *RealExpr1* and *RealExpr2*, respectively. The measurement unit is specified by the *StrExpr* parameter. If null, the default unit is pixels. The *IntExpr2* and *IntExpr2* parameters are reserved for future use.

The coordinate system is consistent with the Mask Editor display that corresponds to the image from an actual or virtual camera. The location of the origin (0,0) of the coordinate system depends on the measurement unit.

If the unit is pixels, the origin of the coordinate system is the lower left corner of the image. X and Y are more positive to the right and up, respectively. If the unit is other than pixels, the origin of the coordinate system is the center of the image. X and Y are more positive to the right and up, and more negative to the left and down, respectively.

Returns the ID of the fiducial as an integer. See also **delfid.**

makemstcalmark(*IntExpr1, IntExpr2, RealExpr1, RealExpr2, StrExpr*)

This command instructs the Mask Editor to create a new calibration marker. The center (X, Y) coordinates are specified by *RealExpr1* and *RealExpr2*, respectively. The measurement unit is specified by the *StrExpr* parameter. If null, the default unit is pixels. *IntExpr1* specifies the type of marker to make zero (0) for Test and one (1) for Reference. The *IntExpr2* parameter is reserved for future use.

The coordinate system is consistent with the Mask Editor display that corresponds to the image from an actual or virtual camera. The location of the origin (0, 0) of the coordinate system depends on the measurement unit. If the unit is pixels, the origin of the coordinate system is the lower left corner of the image. X and Y are more positive to the right and up, respectively. If the unit is other than pixels, the origin of the coordinate system is the center of the image. X and Y are more positive to the right and up, and more negative to the left and down, respectively.

This function is available in all MST applications.

maskdata

Causes MetroPro to apply the current default mask to the current data and replace the current data with the result. Returns no value.

max(*RealExpr, RealExpr*)

Returns the maximum of the two *RealExpr* parameters.

maxreal

Returns the maximum value of a real number. This value prints as “1.#INF.” This value is useful to mark an invalid or undefined real number. See the description for functions **getval** and **setnum**.

measure

Causes MetroPro to perform a measurement. Equivalent to clicking on the MetroPro MEASURE button. Returns no value.

message(*StrExpr, IntExpr, RealExpr*)

Displays a MetroPro message box containing the specified *StrExpr* parameter. Returns no value. The message box is displayed for *RealExpr* seconds. There are three modes selectable by *IntExpr*:

Mode	Description
1	Normal — green title bar
2	Error — red title bar
3	Warning — yellow title bar

metric

Sets **millimeters** as the current linear unit for stage axis positions. (Refer to the **english** statement.)

min(*RealExpr, RealExpr*)

Returns the minimum of the two *RealExpr* parameters.

moveaxis(*StrExpr, IntExpr2, RealExpr, IntExpr3*)
moveaxis(*IntExpr1, IntExpr2, RealExpr, IntExpr3*)

These functions move a programmable stage axis to a specified position.

The axis may be specified as a letter (a one-character string) using *StrExpr* or as an axis number using *IntExpr1*. Axis letters are normally "X," "Y," "P," "R," "T," "Z," "r," "A," or "B." Axis numbers are normally in the range [0,7].

The *IntExpr2* parameter specfies the unit. If the value of *IntExpr2* is negative one (–1), the position is in stepping motor controller counts. Otherwise the position is in the unit appropriate for the axis type.

The position is specified by *RealExpr*.

The *IntExpr3* parameter is a wait-for-stop flag. If *IntExpr3* is zero (0), the function returns without waiting for the motion to complete. Note that this leaves the Motion Controller joysticks disabled. If *IntExpr3* is non-zero, the function returns only after the motion is complete.

These functions return no value.

movemask (*IntExpr, RealExpr1, RealExpr2, StrExpr*)

This function moves a mask figure. Returns no value.

The *IntExpr* parameter specifies the ID of a mask figure obtained via **getmaskid**. If the maskid is zero (0), all figures are affected.

The destination X and Y coordinates are specified by *RealExpr1* and *RealExpr2*, respectively. The measurement unit is specified by *StrExpr*. If null, the default unit is pixels.

The coordinate values are for the center of the minimal enclosing rectangle for the figure. The coordinate system is consistent with the Mask Editor display, which corresponds to the image from an actual or virtual camera. The location of the origin (0,0) of the coordinate system depends on the measurement unit.

If the unit is pixels, the origin (0,0) of the coordinate system is the lower left corner of the image. X and Y are more positive to the right and up, respectively. If the unit is other than pixels, the origin (0,0) of the coordinate system is the center of the image. X and Y are more positive to the right and up, and more negative to the left and down, respectively.

The dimensions of the coordinate system are as described for the **getcamsizx** and **getcamsizy** functions.

movemicrolynx(*RealExpr*)

Moves the Micro LYNX stage to the position specified by *RealExpr* in inches or millimeters.

moverp(*RealExpr1, RealExpr2*)
moverp(*RealExpr1, RealExpr2, IntExpr*)

These functions move the programmable stage Roll and Pitch axes to the position specified by *RealExpr1* and *RealExpr2* in degrees or radians. In the first form, the function returns only after the motion is complete. In the second form, *IntExpr* is a wait-for-stop flag. If *IntExpr* is zero, the function returns without waiting for the motion to complete. Note that this leaves the Motion Controller joysticks disabled. If *IntExpr* is nonzero, the function returns only after the motion is complete. These functions return no value.

movet(*RealExpr*)
movet(*RealExpr, IntExpr*)

These functions move the programmable stage Theta axis to the position specified by *RealExpr* in degrees or radians. In the first form, the function returns only after the motion is complete. In the second form, *IntExpr* is a wait-for-stop flag. If *IntExpr* is zero, the function returns without waiting for the motion to complete. Note that this leaves the Motion Controller joysticks disabled. If *IntExpr* is nonzero, the function returns only after the motion is complete. These functions return no value.

movetur(*IntExpr1*)
movetur(*IntExpr1, IntExpr2*)

This function moves the objective turret on a microscope to the position specified by *IntExpr*. The turret must be motorized for this command to work. For the NewView 6000 instrument, the valid *IntExpr* range is one through six (1—6). For the NewView 5000 instrument, the valid range is one through five (1—5). If *IntExpr2* is nonzero or missing, the function returns only after the movement is completed.

movexy(*RealExpr1, RealExpr2*)
movexy(*RealExpr1, RealExpr2, IntExpr*)

These functions move the programmable stage X and Y axes to the position specified by *RealExpr1* and *RealExpr2* in inches or millimeters. In the first form, the function returns only after the motion is complete. In the second form, *IntExpr* is a wait-for-stop flag. If *IntExpr* is zero (0), the function returns without waiting for the motion to complete. Note that this leaves the Motion Controller joysticks disabled. If *IntExpr* is nonzero, the function returns only after the motion is complete. These functions return no value.

movexyz(*RealExpr1, RealExpr2, RealExpr3*)
movexyz(*RealExpr1, RealExpr2, RealExpr3, IntExpr*)

These functions move the programmable stage X, Y, and Z axes to the position specified by *RealExpr1*, *RealExpr2*, and *RealExpr3* in inches or millimeters. In the first form, the function returns only after the motion is complete. In the second form, *IntExpr* is a wait-for-stop flag. If *IntExpr* is zero, the function returns without waiting for the motion to complete. Note that this leaves the Motion Controller joysticks disabled. If *IntExpr* is nonzero, the function returns only after the motion is complete. These functions return no values.

movez(*RealExpr*)
movez(*RealExpr, IntExpr*)

These functions move the programmable stage z axis to the position specified by *RealExpr* in inches or millimeters. In the first form, the function returns only after the motion is complete. In the second form, *IntExpr* is a wait-for-stop flag. If *IntExpr* is zero (0), the function returns without waiting for the motion to complete. Note that this leaves the Motion Controller joysticks disabled. If *IntExpr* is nonzero, the function returns only after the motion is complete. These functions return no values.

Warning! Always set the Motion Controller Z-stop to protect the microscope objective in case of an inadvertent Z axis movement.

movezoom(*RealExpr*)

On systems with the motorized zoom option, this command will move to the specified image zoom. (*RealExpr* is the value of the desired zoom—e.g. 1, .75, etc.). An error is reported if the system does not have the zoom option, or if the zoom value is not recognized or is not available in the system.

next *RealVar*

This statement is always used with a corresponding for statement. It jumps to the specified instructions. Let Step and Limit be hidden variables associated with this statement and the corresponding for statement.

The Step value is added to *RealVar*. If Step is greater than zero (0)and *RealVar* is less than or equal to Limit, or if Step is less than zero (0)and *RealVar* is greater than or equal to Limit, then execution reverts to the statement following the **for** statement.

Otherwise, execution proceeds with the following statement. This statement may also be used with an *IntVar*.

num(*StrExpr*)

Returns the ASCII numeric value of the first character in the StrExpr.

numtok(*StrExpr1, StrExpr2*)

This function is used when parsing a string. Returns the number of tokens found in *StrExpr1*. *StrExpr2* specifies the characters that act as token separators. Refer to the **postok** function.

off error

This statement cancels the effect of an **on error** statement.

ofmti(*StrExpr*)

Sets the format string used by the **print and output** statements and the **val$()** function for formatting integers. The syntax is the same as for the printf() function in the C programming language. The default format is "%d". Passing a null string restores the default. Returns no value.

ofmtr(*StrExpr*)

Sets the format string used by the **print** and **output** statements and the **val$()** function for formatting real numbers. The syntax is the same as for the printf() function in the C programming language. The default format is "%g". Passing a null string restores the default format. Returns no value.

on error gosub *LineId*

This statement sets up to recover from a subsequent error. The **gosub** statement is executed when a recoverable error occurs. If the **gosub** returns with a **return** statement, execution reverts to the *same line where the error occurred*. If the **gosub** returns with an **error return** statement, execution proceeds with the statement immediately *after the line where the error occurred*.

on error goto *LineId*

This statement sets up to recover from a subsequent error. The **goto** statement is executed when a recoverable error occurs. When **on error goto** is executed, the states of the gosub and for statement nesting are saved. If an error **goto** occurs, the saved states are restored.

on *IntExpr* **gosub** *LineId [, LineId]* ...

This statement branches to one of several specified lines. The *IntExpr* is evaluated and the corresponding *LineId* (1=first, 2=second,...) is used as the destination of a **gosub** statement.

on *IntExpr* **goto** *LineId [, LineId]* ...

This statement branches to one of several specified lines. The *IntExpr* is evaluated and the corresponding *LineId* (1=first, 2=second,...) is used as the destination of a **goto** statement.

openapp(*IntExpr*)

This function opens a closed (iconized) application. Returns no value. The *IntExpr* parameter specifies the ID of the application, obtained via **getappid**. This function does nothing if the application is already open. This function can only be used in a top-level MetroScript.

openwin(*IntExpr*)

This function opens a closed (iconized) window within an application. Returns no value. The *IntExpr* parameter specifies the ID of the window, obtained via **getwinid**. A window opened with this function is not locked even if the application was locked.

output *@FileVar* **;** *Expr* ...

This statement outputs the *Expr* values to the file associated with *FileVar*. *FileVar* must have been created by a previous **assign** statement.

pause

This statement pauses the program and places it in single-step mode. The next statement to be executed is printed on the terminal. Then control returns to the MetroScript prompt. To execute the next one or more statements, use the **continue** command. To leave single-step mode and resume execution from the stopping point, use the **run** command.

pos(*StrExpr1, StrExpr2*)

Returns the position of *StrExpr2* within *StrExpr1*, or zero (0)if not found.

postok(*StrExpr1, StrExpr2, IntExpr*)

This function is used when parsing a string. It returns the character position of the specified token. *StrExpr1* specifies the string to be parsed. *StrExpr2* specifies the characters that act as token separators. *IntExpr* specifies the token number. See also the **numtok** function.

postok, continued-

StrExpr1 = "The,Brown,Dog,Ran" (Separators are the commas.)

StrExpr2 = ","

IntExpr = 0 for "The"
1 for "Brown"
2 for "Dog"
3 for "Ran"

print *Expr [, Expr]* ...
print *Expr [; Expr]* ... ;

Outputs the values of the specified *Exprs*. If the **"printer is"** command was used to specify a filename, output is sent to a text file. Otherwise, in interactive mode, output is sent to the console window. In noninteractive mode, only the first 10 lines of output are accumulated and displayed in a dialog box when the script ends.

The use of commas and semicolons is significant. Comma separators cause values to be output on 14-character field boundaries with at least one blank between fields. Semicolon separators cause values to be output with no added blanks. Normally each print statement outputs one line terminated by carriage-return and line-feed characters. A trailing semicolon causes the line terminators to be omitted so that a subsequent print statement outputs the same line.

A shorthand for **print** is a question mark "?", which is convenient when using the interactive mode to query values of variables.

printer is(*FileNameExpr)*

This statement redirects the output of print statements to a new file with pathname *FileNameExpr*. If *FileNameExpr* is a zero length string, any previous redirection is canceled. This statement allows the "$" character within the file/path name.

printwin(*IntExpr1, IntExpr2, IntExpr3, IntExpr4, IntExpr5, IntExpr6, IntExpr7, FileNameExpr*)

This function is used to print any open MetroPro window with all the options available interactively in the various print dialogs. Options include Source, Destination, Image Format, Data Format, and Range. Either a bitmap or tiff file can be printed with a black (default), or white background.

IntExpr1 is the ID of a window obtained via the **getwinid** function.

IntExpr2 specifies the source (if relevant): zero (0) for image; one (1) for data. If image is specified, the window must be open and completely visible.

IntExpr3 specifies the destination: 0 for printer; 1 for file.

IntExpr4 specifies the output format (if relevant). If the source is a bitmap **image** (.bmp), it is 0. If the source is a .tif image, then it is a 1. If the source is **data** as displayed, it is 1. If the source is data comma separated, it is a 2. If the source is data tab separated, it is a 3.

IntExpr5 and *IntExpr6* specify the range of data items (if relevant). If either *IntExpr5* or *IntExpr6* is 0, then all data items are printed. Otherwise, *IntExpr5* and *IntExpr6* specify the indices for the selected first and last data items respectively.

IntExpr7 specifies whether a heading is to be printed before the data items (if relevant). When set to 1, the image will be printed with a white background. When set to 0, the image will be printed with a black background.

File nameExpr specifies the destination file (if relevant).

promptentry(*IntExpr*)

Displays a MetroPro prompt for the entry of the item referenced by ID *IntExpr*. Returns no value. The ID must reference one of these controls: Lot Num, Part Ser Num, or Auto Save Data File.

promptstr$(*StrExpr1, StrExpr2, IntExpr*)

Displays a MetroPro prompt for the entry of a string. *StrExpr1* is the prompt. *StrExpr2* is the default entry. *IntExpr* is the maximum length of the entry. Returns the entered string.

quit

This command is only usable in the interactive mode. Refer to *Appendix B*, *Interactive Mode*, for information on using this command.

rad

Sets **radians** as the current angle unit for the trigonometric functions and for stage axis positions. (See also the **deg** statement.)

randomize *[RealExpr]*

This statement resets the random number function (**rnd**) with a random seed derived from *RealExpr*. This is useful when identical results are desired from every run of a program using random numbers. If *RealExpr* is omitted, the random seed is derived from the real-time clock in the computer. Returns no value.

read *Var [, Var]* ...

This statement assigns the next unused values in previous **data** statements to the specified *Vars*.

readdmi(*StrExpr*)

This function initiates a continuous display of the position reported by a Displacement Measuring Instrument (DMI). A readout box is displayed on both the console and the video monitor until the operator presses a key or mouse button. This is similar to clicking the Read button in a MetroPro DMI Test data window in the Radius Scale Application.

The *StrExpr* parameter specifies an optional message string. If non-null, the string is displayed in a message box below the readout box. Returns no value.

rem *Remark*

This statement is used for comments. *Remark* is any text. This statement has no effect on the execution of the program. See also the exclamation character "**!**".

renum

This command is only usable in the interactive mode. Refer to *Appendix B*, *Interactive Mode*, for information on using this command.

resaveapp(*IntExpr*)

This function resaves an application. Returns no value. The *IntExpr* parameter specifies the ID of the application, obtained via **getappid**. The application is saved to the same file from which it was last loaded, or to which it was last saved. If the application was not loaded or saved, the default file is “MyApp.app.”

reset

Resets the current MetroPro application. Equivalent to clicking the MetroPro Reset button. Returns no value.

resetdata

Clears any data from MetroPro. Equivalent to clicking on the MetroPro Reset Data button. Returns no value.

resetdmi

Resets and initializes the distance measuring instrument. Returns no value. This function should be called before using functions **setdmimode**, **getdmipos** or **trackdmi**.

resetmasks

Resets MetroPro masks to the default state. Equivalent to clicking the Mask Editor Clear button. Returns no value.

resetmc

Resets the programmable stage Motion Controller. Equivalent to clicking on the MetroPro Reset MC button. Returns no value.

resetmicrolynx

Resets the Micro LYNX. This operation is equivalent to clicking the Reset button in the MetroPro Micro LYNX window.

resizemask(*IntExpr*, *RealExpr*)

This function resizes a mask figure. Returns no value.

The *IntExpr* parameter specifies the ID of a mask figure obtained via **getmaskid**. If the maskid is zero (0), all figures are affected. The new size is specified by *RealExpr* as a percentage. If the resulting size is zero, the mask figure is deleted.

restore

This statement restarts the pointer for **read** statements at the beginning of the available **data** statements.

return

This statement causes execution to proceed to the line saved by the previous **gosub** statement.

rnd

Returns a random number in the range zero (0) to one (1), including 0 but excluding 1.

rotatedata(*RealExpr*)

Causes MetroPro to rotate the current data counterclockwise by the angle specified by the *RealExpr* parameter in the current angle unit. Equivalent to clicking on the MetroPro Rotate Data button. Returns no value.

rotatemask(*IntExpr, RealExpr, StrExpr*)

This function rotates one or all mask figures. Returns no value.

The *IntExpr* parameter specifies the ID of a mask obtained via **getmaskid**. If the ID is zero (0), all figures are affected.

The *RealExpr* parameter specifies the angle.

The *StrExpr* parameter specifies the angle unit, either “deg” or “rad.”

A positive angle rotates counterclockwise; a negative angle rotates clockwise.

run

This command is only usable in the interactive mode. Refer to *Apendix B*, *Interactive Mode*, for information on this command.

runpattern

Causes MetroPro to run the current stage control pattern. Equivalent to clicking on the MetroPro Run Pattern button. Returns no value.

runscript(*IntExpr, FileNameExpr*)

This function runs another script. In this case, the *IntExpr* parameter specifies the ID of an application obtained via the **getappid** function. The function **runscript** can also be called from within an application-level script. In this case the value of *IntExpr* should be zero (0).

The *FileNameExpr* specifies the script file name. Returns no value.

save

This command is only usable in the interactive mode. Refer to *Appendix B*, *Interactive Mode*, for information on using this command.

saveautolatcalinfo(*FileNameExpr*)

This command outputs statistics from the last auto lateral calibration. Refer to OMP-0484 and OMP-0347. A sample output file is shown below.

The *FileNameExpr* parameter specifies the name of the file to create.

```
# Zygo Auto Lateral Calibration Statistics File
# Modified Thu Mar 20 14:26:04 2003
# User "zygo" on host "ZYGO"
#

## System magnification
sys_mag:                9.918764
## Grid size statistics, calibration size
n_cal_rows:             13
n_cal_cols:             13
## Grid size statistics, actual size
n_rows:                 13
n_cols:                 18
## Grid centroid spacing statistics (units: camera pixels)
##    Overall
ovl_mean:               36.190870
ovl_max:                36.915086
ovl_min:                35.629488
##    Horizontal
hor_mean:               36.170838
hor_max:                36.627774
hor_min:                35.879635
##    Vertical
ver_mean:               36.210902
ver_max:                36.915086
ver_min:                35.629488
##
## Grid area statistics (units: camera pixels)
area_mean:              292.260684
      area_sdev:              12.198343
      area_max:               319.018519
      area_min:               255.997230
```

sys_mag: Calculated nominal magnification of the instrument.

n_cal_rows: Number of rows of features used within the lateral calibration routine.

n_cal_cols: Number of columns of features used within the lateral calibration routine.

*n_rows:*Total number of rows detected on the standard.

*n_rows:*Total number of columns detected on the standard.

ovl_mean: Average of all the spacing between features. This result is the measured pitch of the standard in pixels.

ovl_max: Maximum calculated spacing between the centroid of two features.

ovl_min: Minimum calculated spacing between the centroid of two features.

hor_mean: Average of the horizontal spacing between all features. The spacing is calculated as the horizontal distance between the centroid of two features. This result indicates the average spacing between columns of features.

hor_max: Maximum calculated horizontal distance between any two features.

hor_min: Minimum calculated horizontal distance between any two features.

ver_mean: Average of the vertical spacing between all features. The spacing is calculated as the vertical distance between the centroid of two features. This result indicates the average spacing between rows of features.

ver_max: Maximum calculated vertical distance between any two features.

ver_min: Minimum calculated vertical distance between any two features.

area_mean: The mean area size of the isolated data regions found on the standard (usually individual features). Area is expressed as the total number of pixels that make up each individual isolated data regions.

area_sdev: Standard deviation of the area size.

area_max: Maximum area size of any standard dot or other item detected on the standard.

area_min: Minimum area size of any standard dot or other item detected on the standard.

savedata(*IntExpr, FileNameExpr*)

Causes the MetroPro application or data window to save the current data to the file specified by the *FileNameExpr* parameter. If *IntExpr* is zero (0), the save operation is performed by the MetroPro app at the top level. Otherwise *IntExpr* must specify the ID of a data window, obtained via function **getid** (not **getwinid**). Then the save operation is performed in that data window. This function is equivalent to clicking on the MetroPro Save Data button and then selecting *FileNameExpr* as the file name in the dialog box . Returns no value.

saveftpsi(*IntExpr, FileNameExpr*)

Causes MetroPro to save the MST data to the file specified by the *FileNameExpr* parameter.

IntExpr specifies the source of the MST data. For all MST applications besides Homogeneity, the source must be zero (0). When loading or saving data from the Homogeneity application, set *IntExpr* to zero (0) for full cavity, and one (1) for empty cavity data. This function is equivalent to clicking on the SaveAcqData button within an Acquisition Data window and then selecting *FileNameExpr* as the file name in the dialog box. Returns no value. An MST data file contains the raw data needed to calculate height maps. MST data is normally only saved or loaded for diagnostic purposes.

This function is available in the following MST applications: Two Surface, Parallel Surface, Homogeneity, and Custom Cavity.

savemasks(*FileNameExpr*)

This command saves the current masks to a file specified by *FileNameExpr*. Returns no value.

savemstcalmarkers(*FileNameExpr*)

This command saves the calibration markers to a calibration marker file (*.clm). The *FileNameExpr* parameter specifies the filename. It is equivalent to clicking the Save button within the Mask Editor while in the Cal Markers mode and then selecting *FileNameExpr* as the file name in the dialog box. This function does not return a value.

This function is available in all MST applications.

savepatpos(*FileNameExpr*)

Causes the MetroPro Pattern Position Editor to save the current pattern position map to the file specified by *FileNameExpr.* Equivalent to selecting the Save item in the Position Editor pop-up menu and then selecting *FileNameExpr* as the current selection field of the dialog box. Returns no value.

savepatstatus(*FileNameExpr*)

Causes the MetroPro Pattern Position Status window to save the current pattern position status map to the file specified by *FileNameExpr*. Equivalent to clicking on the Zygo icon in the title bar of the Pattern Position Status window(to access the Print Panel) and then selecting 'Data' and 'File' followed by *FileNameExpr* as the current selection field of the dialog box. Returns no value.

savepattern(*FileNameExpr*)

Causes the MetroPro Pattern Editor to save the current pattern to the file specified by the *FileNameExpr* parameter. This function is equivalent to clicking on the Pattern Editor Save button and then selecting *FileNameExpr* as the file name in the dialog box. Returns no value.

saveprocstats(*FileNameExpr*, *IntExpr*)

Causes MetroPro to save the information in all currently open Process Stats windows to the file specified by the *FileNameExpr* parameter. This includes Process Stats as well as Profile Stats. The *IntExpr* parameter specifies the file format as follows:

Format	Description

0	As displayed (columns).
1	Comma separated fields.
2	Tab separated fields.

This function is similar to clicking on the ZYGO button in a Process Stats window, selecting the Data and File buttons in the Print Panel, selecting a Data Format, clicking on Print, and then entering *FileNameExpr* in the Current Selection field of the Save dialog box. Returns no value.

savesettings(*FileNameExpr*)

Causes the MetroPro application to save the current settings to the file specified by *FileNameExpr*. Equivalent to clicking on the MetroPro Save Settings button and then selecting *FileNameExpr* as the file name in the dialog box. Returns no value.

savesubpat(*FileNameExpr*)

Causes the pattern currently in the hidden subpattern holding area to be saved to the file specified by the *FileNameExpr* parameter. There is no equivalent in MetroPro. Returns no value. (Refer to *Appendix D* for additional information on subpattern functions.)

saveswli(*FileNameExpr*)

If there is scanning white light interferometry (SWLI) data available following a measurement, this command saves the data file specified by the *FileNameExpr* parameter. This function is equivalent to clicking on the Save SWLI button and then selecting *FileNameExpr* as the file name in the dialog box. No value is returned. A SWLI data file contains the raw data needed by FDA to calculate a height map.

savetobmp*(IntExpr, FileNameExpr, datatype, [top %], [bottom %])*

This command is used to convert and save the current data to the file specified by the *FileNameExpr* parameter. If *IntExpr* is zero, the save operation is performed by the MetroPro app at the top level. Otherwise, *IntExpr* must specify the ID of a data window, obtained via the **getid** function (not **getwinid**). Then the save operation is performed in that data window. The datatype parameter specifies how to convert the source data as follows:

0 Converts intensity.

1 Converts height data.

2 Converts height data to 128 if data is valid; 0 if it is not data.

The optional parameters, top% and bottom%, specify a percentage of the height data used to define the top and bottom limits. Only valid for datatypes 1 and 2. Default is 1%. Returns no value.

savetovpscam*(IntExpr, datatype, [top%], [bottom%])*

Saves and converts MetroPro data to the VisionPro "virtual" camera. In IntExpr is zero, the save operation is performed by the MetroPro application at the top level. Otherwise, IntExpr must specify the ID of a data window obtained via the getid function (not getwinid). Then, the save operation is performed in that data window. The data type parameter specifies how to convert the source data as follows:

0 Converts intensity

1 Converts height data

2 Converts height data to 128 if data is valid; 0 if it is not data.

The optional parameters, top% and bottom% specify a percentage of the data used to define the top and bottom limits.

scaledata(*RealExpr*)

Causes MetroPro to multiply the current data by the value of the *RealExpr* parameter. Equivalent to clicking on the MetroPro Scale Data button. Returns no value.

scratch

This command is only usable in the interactive mode. Refer to *Appendix B*, *Interactive Mode*, for additional information.

setavmode(*IntExpr1*)
setavmode(*IntExpr1, IntExpr2*)

These functions set the Align/View mode for the GPI and similar instruments. If *IntExpr1* is zero (0), the instrument is set to View mode. If *IntExpr1* is 1, the instrument is set to Align mode. In the second form, *IntExpr2* is a wait-for-stop flag. If *IntExpr2* is zero, the function returns without waiting for the mode change to complete. These functions return no value.

setcalinfo

This function sets system calibration information specified by *StrExpr*, which has the same format as a line in the system calibration file. Returns no value.

setcamaoi(*RealExprl, RealExpr2*)

Sets the camera Area-of-Interest origin. This function is only applicable to instruments using the SMD4M15 camera. The (X,Y) coordinates are specified in pixels by *RealExpr1* and *RealExpr2* respectively.

setcamres(*RealExpr, StrExpr*)

Sets the camera resolution (effective pixel spacing) for subsequent measurements. This is equivalent to using the interactive Lateral Calibrator and entering the resolution. The resolution is specified by the *RealExpr* parameter, and the unit is specified by the *StrExpr* parameter. The specified resolution must be reasonable and consistent with the magnification of the instrument, the current objective, the Camera Mode control, and the Image Zoom control.

setdir(*StrExpr1, StrExpr2*)

This command sets the directory path used by MetroPro to save or load all file types. Returns no value. The file type is specified by *StrExpr1* and must be either a null string or one of the names listed under **getdir$**. If *StrExpr1* is a null string, then the directory path is set for all file types. A directory path can be either of two forms. The first form begins with a drive letter, colon, and backslash characters. The second form specifies a network directory. It begins with double-backslash characters and a computer name (the UNC syntax).

Note that when specifying backslash characters in a string constant, it is necessary to double each one since the first in each pair is interpreted as an escape character. Examples:
"c:\\users\\zygo\\temp" \\\\athena\\c_drive\\data

setdistsys(*RealExpr*)

Sets the Dist Sys value for a Simetra FTP instrument to *RealExpr*. This value is normally specified in the system calibration file.

setdmimode(*IntExpr*, *RealExpr*)

This function sets parameters which affect the operation of a Displacement Measuring Instrument (DMI) that utilizes interferometry, such as a ZYGO ZMI. Returns no value.

The *IntExpr* parameter sets the frequency mode. The valid values are 1 and 2. The default value is 1. The *RealExpr* parameter specifies the index of refraction of air. The default value is 1.0002714.

setdmiscale(*RealExpr*)

Sets the DMI scale factor for a Simetra FTP instrument to *RealExpr*. This value is normally specified in the system calibration file.

setfidpat(*FileNameExpr*, *IntExpr*)

This function specifies a fiducial pattern which will control the subsequent operation of function **findfid** when the mode set by function **setfidthresh** is 2. The *FileNameExpr* parameter specifies a data file containing the centered fiducial pattern. The *IntExpr* parameter is reserved for future use; the value must be zero (0). Returns no value.

setfidsize(*IntExpr1, IntExpr2, RealExpr1, RealExpr2, StrExpr*)

This function sets fiducial size parameters that will control the subsequent operation of the function **findfid** when the mode set by function **setfidthresh** is either zero (0) or one (1). Returns no value.

The minimum and maximum number of valid data points in a fiducial is specified by *IntExpr1* and *IntExpr2*, respectively. These parameters may be set to zero (0) to cause **findfid** to disregard the number of valid data points in candidate regions.

The maximum X and Y dimensions of a fiducial are specified by *RealExpr1* and *RealExpr2*, respectively, in the measurement unit specified by *StrExpr*. If null, the default unit is pixels. These parameters may be set to zero to cause **findfid** to disregard the dimensions of candidate regions.

setfidthresh(*IntExpr, RealExpr1, RealExpr2, StrExpr*)

This function sets fiducial thresholding parameters which will control the subsequent operation of the function **findfid**. Returns no value.

The *IntExpr* parameter specifies the mode of operation: 0, 1, or 2.

Mode 0 causes **findfid** to search for a region of dropouts. The remaining parameters to this function are ignored. The parameters specified by function **setfidsize** are applied. The parameters specified by function **setfidpat** are ignored.

Mode 1 causes **findfid** to search for a region of high values. The remaining parameters establish a cutoff value; data points below the cutoff value are ignored. The *RealExpr1* parameter specifies a "sigma" value. The *RealExpr2* parameter specifies a "cutoff" value in the unit specified by *StrExpr*.

If the sigma value is greater than zero (0), the cutoff passed parameter is ignored and a cutoff value is calculated as follows:

cutoff = mean + (sigma • rms)

where `mean` and `rms` are obtained from a best-fit piston surface to the data. Otherwise, the cutoff passed parameter is used directly. To specify a cutoff value for intensity data, the unit specification must be a null string. The parameters specified by function **setfidsize** are also applied. The parameters specified by function **setfidpat** are ignored.

Mode 2 causes **findfid** to search for a region by using convolution with a pattern specified by function **setfidpat**. The *RealExpr1* parameter specifies the minimum number of valid data points as a percentage of the number of points in the pattern. The remaining parameters to this function and the parameters specified by function **setfidsize** are ignored.

Mode 3 causes **findfid** to search for a region of low values. The remaining parameters establish a cutoff value; data points above the cutoff value are ignored. The *RealExpr1* parameter specifies a “sigma” value. The *RealExpr2* parameter specifies a “cutoff” value in the unit specified by *StrExpr*. If the sigma value is greater than zero, the cutoff passed parameter is ignored and a cutoff value is calculated as follows:

cutoff = mean + (sigma • rms)

where mean and rms are obtained from a best-fit piston surface to the data. Otherwise, the cutoff passed parameter is used directly. To specify a cutoff value for intensity data, the unit specification must be a null string. The parameters specified by function **setfidsize** are also applied. The parameters specified by function **setfidpat** are ignored.

setgpizoom (*RealExpr, IntExpr*)

Sets a GPI encoded zoom lens to the position specified by RealExpr. Returns no value. If *IntExpr* is zero, the position *RealExpr* value is in motor counts. An error is generated if the position is out of range.

Otherwise, if IntExpr is non-zero, the position RealExpr value is a magnification. An error is generated if magnification calibration data does not exist, or if the corresponding motor position is out of range.

(This command applies to GPI or MST instruments with an encoded zoom lens.)

setlight(*RealExpr*)
setlight(*RealExpr, IntExpr1*)
setlight(*RealExpr, IntExpr1, IntExpr2*)

These functions set the instrument light level for viewing and/or measuring. MetroPro provides two settings for the light level control (LLC Options) in the calibration file located in c:\metropro\cfg***.1. A value of 0 indicates that the light levels for viewing and measuring are different, and they can be set separately. A value of 1 indicates that the light levels for viewing and measuring are the same. Returns no value.

IntExpr1	RealExpr	IntExpr2	Description
0 or missing	0 to 100 percent	Ignored	Sets the light level for viewing to *RealExpr.*
1	0 to 100 percent	Ignored	Sets the light level for viewing and measuring to *RealExpr*.
2	Ignored	Ignored	Automatically adjusts the light level for viewing to the highest level with no saturated pixels. This is the same as pressing F5 if the LLC Option in the calibration file is 0.
3	Ignored	Ignored	Automatically adjusts the light level for measuring to the highest level with no saturated pixels. This is the same as pressing Ctrl F5 if the LLC Option in the calibration file is 1.
4	Ignored	0 to 20 step amount	If acquisition mode is set to scan, a scan is run as if performing a measurement and checks the frame for saturation. If there are saturated pixels, the light level is reduced by *IntExpr2*, and the process is repeated until there are not saturated pixels. If *IntExpr2* is missing, a default value of 3 is used. If the acquisition mode is not set to scan, the light level is set using *IntExpr1* = 3. This is the same as pressing Shift F5.

setmcafzero

Instructs the motion controller to save the current position as the auto focus zero position. This command must be called for the motion controller's auto focus feature to be enabled.

setnum(*IntExpr, RealExpr, StrExpr*)

Sets the value of the MetroPro control referenced by ID *IntExpr* to *RealExpr* in the unit specified by *StrExpr*. This function may be used to set the value of numeric, selection, and Boolean type items. Use **setstr** to set the value of string type items and limits. Returns no value.

The operation depends on the item type as follows:

Item Type	Operation
Numeric	The value is set to *RealExpr* in the unit specified by *StrExpr*. Certain numeric controls can be disabled by setting them to blank. To disable a numeric control, use **maxreal** as the *RealExpr*.
Selection	Uses the integer part of *RealExpr* as a selection index (1..N). *StrExpr* is ignored.
Boolean	Sets "Yes" if the value of *RealExpr* is non-zero, otherwise "No". *StrExpr* is ignored.

setpatorg

Sets the pattern origin to the current stage position. Equivalent to clicking on the Pattern Editor Set Origin button. Returns no value.

setpatpos(*RealVar1, RealVar2, IntExpr, StrExpr*)

This function sets positions in a Free Rect type stage control pattern. *Var1* is a one-dimensional real array containing X coordinates. *Var2* is a one-dimensional real array containing Y coordinates. *IntExpr* specifies the number of positions. *StrExpr* specifies the unit. Returns no value.

setpatposstat(*IntExpr1, IntExpr2, StrExpr*)

This function sets the row and column position status for the current pattern. *IntExpr1* specifies the row index. *IntExpr2* specifies the column index. *StrExpr* specifies the status to set the pattern position to. (Refer to "Example 10" in the *Example Scripts* section for a sample script using this function.)

setpatval(*StrExpr1, Expr, StrExpr2*)

This function sets a value in the current stage control pattern. (This is the pattern viewable in the Pattern Editor.) *StrExpr1* specifies the value name. (See the table of pattern values following the **getsubpatval** command.). *Expr* specifies the value to be assigned. *StrExpr2* specifies the unit, if relevant. Returns no value.

setrsmode(*IntExpr1*)
setrsmode(*IntExpr1*, *IntExpr2*)

These functions set the ring/spot mode for the GPI and similar instruments. If *IntExpr1* is 0, the instrument is set to Ring mode. If *IntExpr1* is 1, the instrument is set to Spot mode. In the second form, IntExpr2 is a wait-for-stop flag. If *IntExpr2* is zero (0), the function returns without waiting for the mode change to complete. These functions return no value.

setscancal(*RealExpr*)

Sets the scan calibration factor to *RealExpr*. This value is normally specified in the system calibration file.

setstr(*IntExpr, StrExpr*)

Sets the value of the MetroPro control or limits referenced by ID *IntExpr* to *StrExpr*. This function may be used to set the value of numeric, string, selection, and Boolean type items. The value of numeric type items may also be set using **setnum**. *StrExpr* is used the same way as the values in a MetroPro settings file. Extra blanks and the case of letters are not significant. Units specifications may be abbreviated. Returns no value.

The operation depends on the item type as shown in the following table.

Item Type	Operation
Numeric	The value is set by parsing *StrExpr* which should look like one of these: "1.23 um" or "1.23". A units specification is required if the item has an internal unit. Reasonable unit conversion is performed.
String	The value is set to *StrExpr*.
Selection	The selection is set to match *StrExpr*.
Boolean	The value is set using *StrExpr* which must be one of: "On", "Off", "Yes", "No", "True", or "False".
Limits	The limits are set by parsing *StrExpr* which should look like one of these: "[1.23,4.56] um", "[*,*] um", "1.23,4.56 um", "1.23 4.56 um", or "1.23 4.56". The order of the numbers is low then high limit. The brackets and comma are optional. An asterisk in place of a number disables that limit. A units specification is required if the associated control provides the Units menu selection. Reasonable unit conversion is performed.

setsubpatval(*StrExpr1, Expr, StrExpr2*)

This function sets a value in the hidden subpattern holding area. *StrExpr1* specifies the value name. (See the table of pattern values following the **getsubpatval** command). *Expr* specifies the value to be assigned. *StrExpr2* specifies the unit, if relevant. Returns no value. (Refer to *Appendix D* for additional information on subpatterns.)

sgn(*RealExpr*)

Returns -1, 0, or +1 depending on whether *RealExpr* is less than, equal to, or greater than zero.

sherlockcmd*(StrExpr)*

The string parameter is passed to the Surveyor/Sherlock software and is executed as a **lightsout** command. This command returns the string results of the Surveyor/Sherlock software command. (Refer to the *Sherlock User's Reference* manual for a complete list of **lightsout** commands.) If the string parameter is "shutdown," it causes Surveyor/Sherlock to shutdown.

showcalmarkers

This command causes the mask editor to switch to the calibration markers mode.

This function is available in all MST applications.

showfids

Opens the mask editor and displays the fiducials.

showmasks

Displays the mask editor.

sin(*RealExpr*)

Returns the sine of the *RealExpr* parameter.

sinh(*RealExpr*)

Returns the hyperbolic sine of the *RealExpr* parameter.

sqr(*RealExpr*)
sqrt(*RealExpr*)

Both forms return the square root of the *RealExpr* parameter.

step

This command is only usable in the interactive mode. Refer to *Appendix B, Interactive Mode*, for information on using this command.

stop *[Expr]*

This statement halts program execution. The value of the optional *Expr* is returned to MetroPro. *Expr* may be numeric or string type. The **stop** statement without *Expr* is equivalent to the **end** statement.

storeprocstats

Causes MetroPro to store the current results into the process stats. This is equivalent to clicking on the Store button in a Process Stats window. Returns no value.

strerror$(*IntExpr*)

Returns a string describing the system error specified by *IntExpr*. This can be used to interpret a value returned by the **execute** function.

strrepl$(*StrExpr1, StrExpr2, StrExpr3*)

This function performs a string replacement operation. It looks for the first occurrence of *StrExpr2* in *StrExpr1* and replaces it with *StrExpr3*. It returns the result. An error is generated if *StrExpr2* is not found in *StrExpr1*.

subtractdata(*StrExpr*)

Causes MetroPro to subtract from the current data the data file specified by the *StrExpr* parameter. Equivalent to clicking on the MetroPro Subtract Data button. Returns no value.

tan(*RealExpr*)

Returns the tangent of the *RealExpr* parameter.

tanh(*RealExpr*)

Returns the hyperbolic tangent of the *RealExpr* parameter.

then

Refer to the **if** command.

time$(*IntExpr*)

Returns a string containing the time corresponding to *IntExpr* number of seconds. The format is HH:MM:SS. If *IntExpr* is greater than the number of seconds in one year, the result will include the usual correction for time zones and daylight saving.

timedate

Returns the time in seconds since January 1, 1970. This may be used to time events or may be converted to time and date strings using the **time$()** and **date$()** functions.

tobackwin(*IntExpr*)

This function pushes an open window back within an application. This means that the window will be displayed behind any other intersecting windows. The *IntExpr* parameter specifies the ID of the window obtained via **getwinid**. An error is generated if the window is not open. Returns no value.

tofrontwin(*IntExpr*)

This function brings to the front an open window within an application. This means that the window will be displayed on top of any other intersecting windows. The *IntExpr* parameter specifies the ID of the window, obtained via **getwinid**. An error is generated if the window is not open. Returns no value.

translatedata(*StrExpr*)

Translates the current data set in the camera coordinate system. The delta coordinate pair is specified as a string containing two numbers separated by blank, comma, or tab characters.

trim$(*StrExpr*)

Returns a copy of *StrExpr* without leading or trailing spaces.

trimdata(*RealExpr*)

Causes MetroPro to trim the current data by the number of pixels specified by the *RealExpr* parameter . Returns no value.

unclampstage(*IntExpr*)

Causes the motion controller to unclamp a single stage or all stages as indicated by *IntExpr*. The stage must have clamping enabled in the motor configuration file for this command to have any effect. For unclamping a single stage, *IntExpr* has a valid range of zero (0) to 7. To unclamp all stages, set *IntExpr* to -1.

unlockgpizoom

Unlocks the position of a GPI encoded zoom lens so that it can be changed.

(This command applies to GPI or MST instruments with an encoded zoom lens.)

upc$(*StrExpr*)

Returns a copy of *StrExpr* with all lower case letters converted to upper case.

val(*StrExpr*)

Returns the numeric value of the leading part of *StrExpr*. Returns zero (0) if the leading part of *StrExpr* is non-numeric.

val$(*IntExpr*)
val$(*RealExpr*)

This function returns the string obtained by formatting the value of *IntExpr* or *RealExpr* using the format determined by function **ofmti** or **ofmtr**, respectively.

viewpatstatus(*Real Expr*)

Opens the MetroPro Pattern Position Status window and allows user interaction. Completes when the user closes the window, or presses the ESC key. Pressing the ESC key produces the "Processing aborted" error. Equivalent to clicking on the MetroPro View Pos button.

If *RealExpr* is a1 or larger, then the script will wait for the status window to close.

If *RealExpr* is zero (0), then the script will not wait for the window to close.

vpscmd(*StrExpr*)

Issues a VisionPro Server command. The string parameter is passed to the VisionPro Server and is executed. Results of the command are returned as a string. The following commands are used with this function. All parameters are case sensitive.

- **RT [ToolName]**

Runs the tool with the specified name. If a name is not specified, the entire tool group is run. Tool names cannot contain any whitespace.

- **LT *ToolGroupName***

  Loads the tool group specified by ToolGroupName.

- **SI *InputPath Value***

  Sets the tool group input specified by InputPath to the value specified by Value.

- QO OutputPath

  Queries the tool group output specified by Output Path.

vpsconnect

Connects to the VisionPro Server application. It starts the application if it isn't already running.

vpsdisconnect

Disconnects from the VisionPro Server application.

wait *RealExpr*

This statement suspends program execution for *RealExpr* seconds.

waitrp

Waits for movement of the programmable stage Roll and Pitch axes to complete. Returns no value. This function is normally called after calling the **movet** function with the wait-for-stop flag set to zero (0).

waitt

Waits for movement of the programmable stage Theta axis to complete. Returns no value. This function is normally called after calling the **movet** function with the wait-for-stop flag set to zero (0).

waitxy

Waits for movement of the programmable stage X and Y axes to complete. Returns no value. This function is normally called after calling the **movexy** function with the wait-for-stop flag set to zero (0).

waitxyz

Waits for movement of the programmable stage X, Y, and Z axes to complete. Returns no value. This function is normally called after calling the **movexyz** function with the wait-for-stop flag set to zero (0).

waitz

Waits for movement of the programmable stage Z axis to complete. Returns no value. This function is normally called after calling the **movez** function with the wait-for-stop flag set to zero (0).

wandoff

Disables the CAN wand buttons.

wandon

Enables the CAN wand buttons.

writecalfile(*FileNameExpr*)

Writes the system calibration file for the instrument to *FileNameExpr*. If *FileNameExpr* is a null string, this function overwrites the startup system calibration file that normally resides in directory c:\metropro\cfg. Returns no value.

zerogage

Performs a zero gage operation for the Simetra FTP instrument. Equivalent to clicking on the zero gage button in an FTP application. Returns no value.

Appendix A
Functional Cross-Reference

Many MetroScript commands are used specifically for control of instruments or for manipulating data acquired with an instrument. These commands are associated with MetroPro functions. Other commands within MetroScript reflect programming functions traditionally found in most programming languages. Therefore, this reference section is divided into two categories: MetroScript Language Commands and MetroPro Related Commands. Within each of these two categories, commands are grouped according to similar functionality.

Generic Language Commands

Math

abs(*RealExpr*)
acs(*RealExpr*)
asn(*RealExpr*)
atn(*RealExpr*)
atn*(RealExpr)*
atn2*(RealExpr1, RealExpr2)*
cos(*RealExpr*)
cosh(*RealExpr*)
deg
exp(*RealExpr*)
int(*RealExpr*)
lgt(*RealExpr*)
log(*RealExpr*)
log2(*RealExpr*)
max(*RealExpr*, *RealExpr*)
maxreal
min(*RealExpr*, *RealExpr*)
rad
randomize *[RealExpr]*
rnd
sgn(*RealExpr*)
sin(*RealExpr*)
sinh(*RealExpr*)
sqr(*RealExpr*)
sqrt(*RealExpr*)
tan(*RealExpr*)
tanh(*RealExpr*)

Program Control

dim *DimItem [, DimItem] …*
for *RealVar = RealExpr1* to
RealExpr2 [step *RealExpr3]*
end
execute(*StrExpr*)
exitmp(*IntExpr*)
gosub *LineId*
goto *LineId*
if *LogicExpr* then *Stmt1*
[else *Stmt2]*
if *LogicExpr* then
[else]
endif
next *RealVar*
on *IntExpr* gosub *LineId*
[, LineId] ...
on *IntExpr* goto *LineId*
[, LineId] ...
pause
rem *Remark*
return
runscript(*IntExpr, FileNameExpr*)
stop [*Expr*]
wait *RealExpr*

Generic Language Commands, continued-

File and Serial Port Input/Output

assign *@FileVar* to *FileNameExpr [StrConst]*
disable abort *@FileVar*
enable abort *@FileVar*
enter *@FileVar ; Var [, Var] ...*
enterline *@FileVar; StrVar*
output *@FileVar ; Expr ...*
logfile is *FileNameExpr*
print *Expr [, Expr] ...*
print *Expr [; Expr] ... ;*
printer is *FileNameExpr*
printwin*(IntExpr1, IntExpr2, IntExpr3, IntExpr4, IntExpr5, IntExpr6, IntExpr7, FileNameExpr)*

Error Handling

errl(*LineId*)
errln
errm$
errn
error return
on error gosub *LineId*
on error goto *LineId*
off error

Program Data

data *Const [, Const] ...*
read *Var [, Var] ...*
restore

String Manipulation

chr$(*RealExpr*)
date$(*RealExpr*)
len(*StrExpr*)
lwc$(*StrExpr*)
num(*StrExpr*)
numtok*(StrExpr1, StrExpr2)*
ofmti(*StrExpr*)
ofmtr(*StrExpr*)
pos(*StrExpr1, StrExpr2*)
postok*(StrExpr1, StrExpr2, IntExpr)*
strerror$*(IntExpr)*
strrepl*$(StrExpr1, StrExpr2, StrExpr3)*
StrVar [StartIndex, EndIndex]
StrVar [StartIndex; Length]
StrVar [StartIndex]
time$(*IntExpr*)
timedate
trim$(*StrExpr*)
upc$(*StrExpr*)
val$(*RealExpr*)
val(*StrExpr*)

Debugging Commands (*Interactive Mode*)

break
continue
del
edit
exit
input *StrConst , Var [, Var] ...*
input *Var [, Var]*
load
loadsub *FileNameConst*
print *Expr [, Expr] ...*
printer is *FileNameExpr*
renum
run
save
scratch
step

MetroPro Related Commands

Interaction with MetroPro Objects

clickon(*IntExpr*)
closeapp(*IntExpr*)
closewin(*IntExpr*)
getannotid(*IntExpr, StrExpr*)
getappid(*StrExpr*)
getid(*StrExpr*)
getnum(*IntExpr, StrExpr*)
getstr$(*StrExpr*)
getval$(*IntExpr*)
getval(*IntExpr, StrExpr*)
getmaskid(*RealExpr1, RealExpr2, StrExpr*)
getwinid(*StrExpr*)
loadapp(*StrExpr*)
openapp(*StrExpr*)
openwin(*IntExpr*)
resaveapp*(IntExpr)*
setnum(*IntExpr, RealExpr, StrExpr*)
setstr(*IntExpr, StrExpr*)
sherlockcmd*(StrExpr)*
tobackwin(*IntExpr*)
tofrontwin(*IntExpr*)

Motion/Stage Control

autofocus
autofocustilt
autotilt
english
getppos
getrpos
gettpos
getxpos
getypos
getzpos
homeaxis*(StrExpr)*
homeaxis*(IntExpr)*
homestage
isaxisactive*(StrExpr)*
isaxisactive*(IntExpr)*
joyoff
joyon
logreports
metric
moveaxis*(StrExpr, IntExpr1, RealExpr, IntExpr2)*
moveaxis*(IntExpr1, IntExpr2, RealExpr, IntExpr3)*
moverp(*RealExpr1, RealExpr2* (*RealExpr1, RealExpr2, IntExp*r)
movet*(RealExpr)*
movet*(RealExpr, IntExpr)*
movexy(*RealExpr1, RealExpr2*) (*RealExpr1, RealExpr2, IntExpr*)
movez(*RealExpr*) (*RealExpr, IntExpr*)
resetmc
waitt
waitxy
waitrp
waitz

MetroPro Related Commands, continued-

Displacement Measuring Instrument

getdmipos(*StrExpr*)

Directories and Operating Systems

getbindir$

getdmiscale
getdmitype$
readdmi(*StrExpr*)
resetdmi
setdmimode(*IntExpr, RealExpr*)
setdmiscale*(RealExpr)*

Reports

logreports

Settings Files

loadpartconfig(*FileNameExpr*)
loadsettings(*FileNameExpr*)
savesettings(*FileNameExpr*)

Fiducials

delfid(*IntExpr*)
findfid(*IntExpr, RealExpr1, RealExpr2, RealExpr3, RealExpr4, StrExpr*)
getfidctrx(*IntExpr, StrExpr*)
getfidctry(*IntExpr, StrExpr*)
getfidorgx(*IntExpr, StrExpr*)
getfidorgy(*IntExpr, StrExpr*)
getfidsizx(*IntExpr, StrExpr*)
getfidsizy(*IntExpr, StrExpr*)
setfidpat*(FileNameExpr, IntExpr*)
setfidsize(*IntExpr1, IntExpr2, RealExpr1, RealExpr2, StrExpr*)
makefid(*RealExpr1, RealExpr2, StrExpr, IntExpr1, IntExpr2*)
setfidthresh(*IntExpr, RealExpr1, RealExpr2, StrExpr*)
showfids

getdir$(*StrExpr*)
getdiskfree(*StrExpr*)
getosname$
getramsize
getwrkdir$
setdir(*StrExpr1, StrExpr2*)

Loading/Saving Data

loadbitmap(*IntExpr, StrExpr*)
saveautolatcalinfo*(FileNameExpr)*
savetobmp*(IntExpr, FileNameExpr, datatype, [top%], [bottom %])*

Process Data

clearprocstats
saveprocstats(*StrExpr, IntExpr*)
storeprocstats

Masks

adjustmask(*IntExpr, RealExpr1, RealExpr2, StrExpr*)
editmasks*(IntExpr*)
getmaskctrx(*IntExpr, StrExpr*)
getmaskctry(*IntExpr, StrExpr*)
getmaskid(*RealExpr1, RealExpr2, StrExpr*)
getmasksizx(*IntExpr, StrE*xpr)
getmasksizy(*IntExpr, StrExpr*)
loadmasks(*FileNameExpr*)
maskdata
movemask(*IntExpr, RealExpr1, RealExpr2, StrExpr*)
resetmasks
resizemask(*IntExpr, RealExpr*)
savemasks(*FileNameExpr*)
showmasks

MetroPro Related Commands, continued-

Instrument

acquire
analyze
autofocus

Stage Control Patterns

contpattern
editpatpos
editpattern

autofocustilt
autolatcal(*IntExpr, RealExpr, StrExpr*)
autotilt
gensyschar(*FileNameExpr*)
gensyschar(*FileNameExpr, SwliFileNameExpr*)
getavmode
getlight
getrsmode
getscancal
getsyserrfn$
getturpos
getzoom
homezoom
llcdialog
measure
movetur(*IntExpr*)
movezoom(*RealExpr*)
reset
setavmode(*IntExpr 1*) (*IntExpr1, IntExpr2*)
setcalinfo
setcamaio (*RealExpr1, RealExpr2*)
setcamres(*RealExpr, StrExpr*)
setdistsys*(RealExpr)*
setlight(*RealExpr*)
setrsmode*(IntExpr1) (IntExpr1, IntExpr2)*
setscancal*(RealExpr)*
writecalfile*(FileNameExpr)*
zerogage

getpatval(*StrExp1, Var, StrExpr2*)
getsubpatval(*StrExp1, Var, StrExpr2*)
gotopatorg
gotopatpos(*IntExpr*)
loadpatpos(*FileNameExpr*)
loadpatstatus (*FileNameExpr*)
loadpattern(*FileNameExpr*)
loadsubpat(*FileNameExpr*)
runpattern
savepatpos(*FileNameExpr*)
savepatstatus(*FileNameExpr*)
savepattern(*FileNameExpr*)
savesubpat(*FileNameExpr*)
setpatposstat(*IntExpr, IntExpr, StrExpr*)
setpatorg
setpatpos(*Var1, Var2, Expr1, Expr2*)
setpatval(*StrExpr1, Expr, StrExpr2*)
setsubpatval(*StrExpr1, Expr, StrExpr2*)
viewpatstatus(*RealExpr*)

MetroPro Related Commands, continued--

Data Manipulation

adddata(*StrExpr*)
convinten
flipdatax
flipdatay

Camera and Data Attributes

getcamres(*StrExpr*)
getcamsiz(*StrExpr*)
getcamsizy(*StrExpr*)
getdatactrx(*IntExpr, StrExpr*)

generate
gensyschar(*FileNameExpr*)
gensyschar(*FileNameExpr, SwliFileNameExpr*)
getzernikes
invertdata
resetdata
rotatedata(*RealExpr*)
savedata(*FileNameExpr*)
scaledata(*RealExpr*)
subtractdata(*StrExpr*)
translatedata(*StrExpr*)
trimdata(*RealExpr*)

Messaging/Prompting

dialog(*StrExpr, IntExpr*)
message(*StrExpr, IntExpr, RealExpr*)
promptentry(*IntExpr*)
promptstr$(*StrExpr1, StrExpr2*, *IntExpr*)

getdatactry(*IntExpr, StrExpr*)
getdataorgx(*IntExpr, StrExpr*)
getdataorgy(*IntExpr, StrExpr*)
getdatasizx(*IntExpr, StrExpr*)
getdatsizy(*IntExpr, StrExpr*)

Turret Control

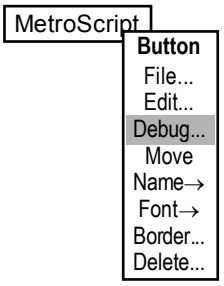
getturpos(*IntExpr*)
movetur(*IntExpr*)

Data Files

autosavedata
loaddata(*FileNameExpr*)
loadswli(*FileNameExpr*)
savedata(*IntExpr, FileNameExpr*)
saveswli(*FileNameExpr*)

Appendix B
Interactive Mode

MetroScript interactive mode is used to test, debug, and edit script files. Script files can be executed a line at a time, helping you to eliminate bugs and problems in your script file. This mode is especially useful for long scripts.

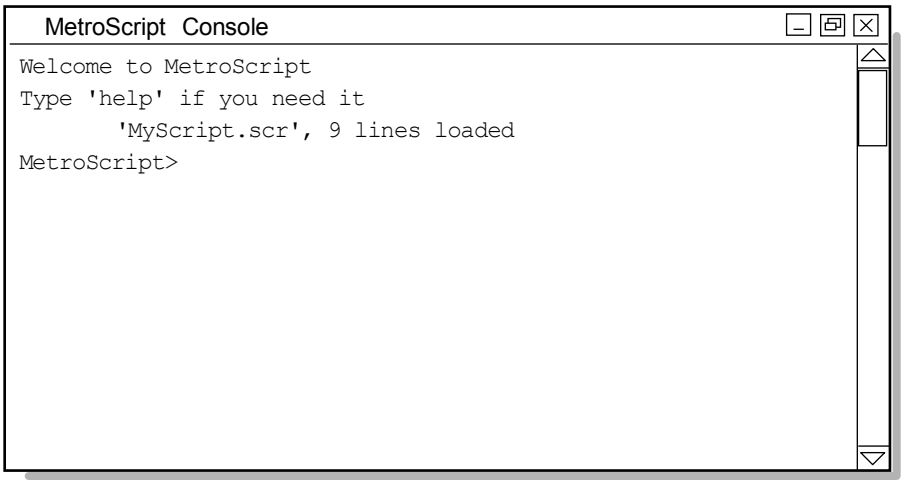


Entering the Interactive Mode

Select the MetroScript button's menu Debug command. A console window will pop up.

Working in the Interactive Mode

When the Debug command is selected, the script file linked to the Metro-Script button is automatically loaded. Lines with a leading tab are assigned line numbers at this time.



Make sure the console window does not cover the center of the screen if your script uses the **promptstr$** or **promptentry** functions.

Enter commands at the MetroScript prompt. The most common commands are:

Command	Function
`run`	Execute the script file; like clicking the MetroScript button.
`step`	Enters the single-step mode for executing the script one line at a time. Press [enter] to execute the next line. Type **c** *Num* to execute that number of lines.
`list`	Displays on screen the number of lines specified. List 10-20 displays lines 10 through 20.

Interactive Mode Command Listing

This section describes MetroScript commands that are used in the interactive mode. Most of the commands described in this section are usable *only* at the MetroScript prompt; they are *not*

allowed within a MetroScript program. Any command can be abbreviated as long as it is unambiguous. For example, "q" is an acceptable form for **quit**, "c" for **continue**, and "st" for **step**.

Almost any MetroScript statement that could appear in a program can be used interactively. That is, the statement can be entered at the MetroScript prompt without a line number or leading *Tab* character to cause it to be executed immediately.

! [*SysCmd*]

A command line beginning with an exclamation mark provides immediate access to operating system commands (without opening another console window).

The optional *SysCmd* is anything that would be typed at a command (or 'shell') prompt.

Under WindowsNT, *SysCmd* is attempted to be executed first as an 'external' program. If that fails, *SysCmd* is passed to the WindowsNT command interpreter.

After executing *SysCmd*, the MetroScript prompt is restored.

If no *SysCmd* is specified, a command interpreter is invoked that takes control of the console window.

The command prompt is displayed and any operating system commands may be entered. Type **exit** to return to the MetroScript prompt.

LineNum [*LineLabel*] [*Stmt*] [! *Comment*]

Enters a program line into the workspace with a fixed line number *LineNum*. If a line with the same *LineNum* already exists in the workspace, it is deleted.

Tab [*LineLabel*] [*Stmt*] [! *Comment*]

Enters a program line into the workspace with automatic line numbering. The line will be given a number 10 higher than the highest existing line number, rounded off to a multiple of 10. Mixing fixed and automatically numbered lines in the same program file is undesirable because lines may be unexpectedly deleted or placed in a different order. When any program is saved using the **save** command, line numbers are included.

break *LineId*

Sets a breakpoint to cause the program to enter single-step mode when the line corresponding to *LineId* is about to be executed.

continue [*IntConst*]

In single-step mode, the next *IntConst* statements are executed. If *IntConst* is omitted, one statement is executed. Note that **continue** can be abbreviated "**c**". In single-step mode, entering a blank line is equivalent to **continue 1**.

del *LineNumRange*

Deletes the specified lines from the workspace.

edit

This command runs an editor on the contents of the workspace (with line numbers).
The editor program is **NotePad**. After the editor is exited, the result is loaded back into the workspace. This is one way to perform non-trivial program editing. See also the "Writing Script Files" section in *Chapter 2*.

exit

This command exits Interactive Mode.

help

Outputs a list of the essential interactive mode commands.

gosub
goto

The **gosub** and **goto** statements (described in Chapter 4) are normally used within programs. They are also useful as interactive commands. They can be used to initiate execution of statements without resetting program variables (in contrast to the **run** command). They are also useful in conjunction with the **step** command.

list [*LineNumRange*]

Outputs all or specified program lines to the display.

load [*FileNameConst*]

The workspace is cleared and the specified file is loaded into the workspace. If the file contains embedded commands (lines without a line number or leading *Tab*), they are executed as they are read and not placed in the workspace. The workspace is assigned the name *FileNameConst.* If *FileNameConst* is omitted, the workspace is cleared and the file last saved or loaded is loaded back into the workspace.

loadsub *FileNameConst*

The specified file is loaded into the workspace at the end of the existing program. The line numbers of the new lines are adjusted to begin at the next 1000 line boundary. This command allows the "$" character within the file/path name.

quit

Exit MetroScript and return to MetroPro (same as **exit**).

renum [*LineNum1*] [**in** *LineNumRange*]

Renumbers program lines. If *LineNum1* is specified, the renumbered lines will start with *LineNum1* if specified otherwise 100. If **in** *LineNumRange* is specified, only those lines are renumbered. Otherwise, all program lines are renumbered. Corresponding line numbers referenced in **goto**, **gosub**, **if**, and **on** statements are adjusted. But the renumbering process cannot adjust integer arguments to the **errl()** function or integers compared to the returned value from the **errln** function.

run

The workspace variables are cleared and the program in the workspace is run. If the program was in single-step mode, **run** resumes execution from the stopping point in normal run mode.

run loop *N*

Repeats the **run** command N times.

save [*FileNameConst*]

Saves the current program to the specified file. If the workspace had no name, it becomes *FileNameConst*. If *FileNameConst* is omitted, the current workspace name is used as a default. An exception is if the loaded file contained commands, "**!**" shell escapes, or "**#**" comments, or if a **loadsub** was done; in these cases, the default filename usage is prevented to avoid unintended results.

scratch

Clears the workspace program and variables.

step

Begin single-step mode. A **goto** or **gosub** command can be entered to change the **next** statement to be executed. Single-step mode remains in effect until program execution is halted by a **stop** or **end** statement, either in the program or entered at the MetroScript prompt.

stop (or **end**)

The **stop** and **end** statements (described in *Chapter 4*) are normally used within programs. In interactive mode, use **stop** or **end** to terminate the single-step mode (see the **step** command).

Appendix C
Available Controls

Controls in the GPI Application

The following table is a list of control boxes found in the Control window in the basic GPI application. They are listed to help you find a less-used or unused control that can be taken over when writing a script file. Refer to *Chapter 2* for more information on using input from Control boxes.

The Off Limits and Okay columns identify some of the instruments and/or conditions involved in using the control. "All" means that it is either off limits (or okay) for all instruments. "All others" means that other conditions are listed. The Units column identifies the units used with the control if you are looking for a particular type of control box, such as Numeric, alpha, or on/off.

Note: MetroPro now includes a Custom Result option. It can be used to output a measurement result that is customized for a specific measurement. A Custom result may be Boolean (on/off), numeric, or text. Refer to the "Results" section of the *MetroPro Reference Guide*, OMP-0347, for additional information. The controls listed as Okay in this section can still be used as well.

Sub-menu/Control	Off Limits	Okay	Units	Comments
Interferometer				
Instrument	All			
Camera Mode	All			
Image Zoom	All			
Intf Scale Factor	All			
Wavelength-In	All			
Acquisition				
Acquisition Mode	All			
Intens Avgs	All			
AGC	All others	SWLI microscopes	Y/N	
AGC Mode	All others	SWLI microscopes	List	Not particularly useful
Light Level	All			
Light Level Pct	All			
LLC Delay	All			0-10
Target Range	All others	SWLI microscopes	Numeric	0-100
Max Sat Pts	All others	SWLI microscopes	Numeric	>=0
Ignore Bright Pts	All others	SWLI microscopes	Y/N	
PZT Cal	All others	SWLI microscopes	Y/N	
PZT Gain	All others	SWLI microscopes and GPI in Intens Acquisition Mode	Numeric	0 - 4095
PZT Scale Factor	All others	SWLI microscopes and GPI in Intens Acquisition Mode	Numeric	0.125 - 2.0
Min Mod (%)	All			
Min Mod Pts	All			
Remove Fringes	All			
Phase Processing				
Fringe Threshold	GPI for fringe analysis	All others	Numeric	0 - 1
Fringe Limit	GPI for fringe analysis	All others	Numeric	0 - 10
Fringe Spacing	GPI for fringe analysis	All others	Numeric	0 - 1
Phase Processing				
Trim	All			
Trim Mode	All			
Phase Avgs	All			
Phase Avg Pause	All			
Phase Res	All			
Min Area Size	All			
Max Area Size	All			
Pre-Connect Filter	All			
Connection Type	All			
Connection Order	All			
Discon Action	All			
Discon Filter	All			
Remove Tilt Bias	All			
Remove	All			
Data Sign	All			

Controls in the GPI Application (continued)

Generate				
Surface Type		Except when generating	List	Limited usefulness
Camera Size X		Except when generating	Numeric	50 - 10500
Camera Size Y		Except when generating	Numeric	50 - 10500
Camera Res		Except when generating	lateral	>= 0
Generate/Zernike				
Zernike Terms		Except when generating	List	Limited usefulness
Center X		Except when generating	Numeric	
Center Y		Except when generating	Numeric	
Radius		Except when generating	Numeric	
Coefs 0-15		Except when generating	Height	Scaled by Wavelength-In and Wavelength-Out, number will change if either of these values change.
Coefs 16-35		Except when generating	Height	Same as Coefs 0-15.
System Error				
Subtract Sys Err	All			
Sys Err File	If Subtract Sys Err On	If Subtract Sys Err Off	Alpha	Limit of 14 characters
Manipulate				
Alignment	If adding/subtracting data sets	All others	Y/N	
Alignment Tol	If using Fiducials	All others	Numeric	>=0
Alignment Scaling	If using Fiducials	All others	List	Limited usefulness
Auto Sequence				
Auto Seq Operation	If using Auto Sequence	All others	List	Limited usefulness
Auto Save Data	If using Auto Sequence	All others	Y/N	
Auto Save Data File	If Auto Save Data On	If Auto Save Data Off	Alpha	Limit of 14 characters
Auto Seq Max Count	If using Auto Sequence	All others	Numeric	> 0
Auto Seq Delay	If using Auto Sequence	All others	Numeric	> 0
Auto Seq Interval	If using Auto Sequence	All others	Numeric	> 0
Auto Tilt				
Auto Tilt	All			
Auto Tilt Mode	If Auto Tilt On	All others	List	Limited usefulness
Auto Tilt Roll Offset	If Auto Tilt On	All others	Numeric	-2 - 2 degrees
Auto Tilt Pitch Offset	If Auto Tilt On	All others	Numeric	-2 - 2 degrees
Auto Tilt Roll Tol	If Auto Tilt On	All others	Numeric	0 - 2 degrees
Auto Tilt Pitch Tol	If Auto Tilt On	All others	Numeric	0 - 2 degrees
Auto Tilt Min Contrast	If Auto Tilt On	All others	Numeric	0 - 100
Auto Tilt Domain	If Auto Tilt On	All others	Numeric	0.1 - 2 degrees
Auto Tilt Max Adjust	If Auto Tilt On	All others	Numeric	0 - 4 degrees
Environmental				
Temperature (C)	If using IRS	All others	Numeric	-99 - 99
Pressure (Torr)	If using IRS	All others	Numeric	0 - 99999
Relative Humidity (%)	If using IRS	All others	Numeric	0 - 100
Index of Air			Attribute	Cannot be used
Masks				

Show Masks	All			
Auto Load Masks	All			
Masks File	If Auto Load Masks On	If Auto Load Masks Off	Alpha	Limit of 14 characters

Controls in the GPI Application (continued)

Pattern				
Auto Load Pattern			All	
Pattern File	If Auto Load Pattern On	If Auto Load Pattern Off	Alpha	Limit of 14 characters
Pattern Number	If using pattern function	All others	Numeric	0 - 99
Stitch				
Type	If using stitching	All others	List	Limited usefulness
Display Measurements	If using stitching	All others	Y/N	
Auto Save Seq	If using stitching	All others	Y/N	
Seq File	If using stitching	All others	Alpha	Limit of 14 characters
Failure Action	If using stitching	All others	List	Limited usefulness
Auto Focus	If using stitching	All others	Y/N	
Store Process Stats	If using stitching	All others	Y/N	
Scanning Seq	If using stitching	All others	List	Limited usefulness
Stitch Processed Data	If using stitching	All others	Y/N	
Average Overlap Regions	If using stitching	All others	Y/N	
Diagnostics			Y/N	
Auto Load Masks	If using stitching	All others	Y/N	
Masks File	If using stitching	All others	Alpha	Limit of 14 characters
N Cols	If using stitching	All others	Numeric	1 - 50
N Rows	If using stitching	All others	Numeric	1 - 50
Overlap (%)	If using stitching	All others	Numeric	10 - 90
Size X	If using stitching	All others	Linear	>= 0
Size Y	If using stitching	All others	Linear	>= 0
Zip Stitch				
Zip Distance	If using stitching	All others	Linear	>= 0
Auto Load Masks	If using stitching	All others	Y/N	
Masks File	If using stitching	All others	Alpha	Limit of 14 characters
Store Prrocess Stats	If using stitching	All others	Y/N	
Lower Overlap (%)	If using stitching	All others	Numeric	10 - 90
Zip Stitch				
Lower Size X	If using stitching	All others	Linear	>= 0
Lower Size Y	If using stitching	All others	Linear	>= 0
Use Lower Seq File	If using stitching	All others	Y/N	
Lower Seq File	If using stitching	All others	Alpha	Limit of 14 characters
Upper Overlap (%)	If using stitching	All others	Numeric	10 - 90
Upper Size X	If using stitching	All others	Linear	>= 0
Upper Size Y	If using stitching	All others	Linear	>= 0
Use Upper Seq File	If using stitching	All others	Y/N	
Upper Seq File	If using stitching	All others	Alpha	Limit of 14 characters
Upper Offset X	If using stitching	All others	Linear	
Upper Offset Y	If using stitching	All others	Linear	
Script				
Auto Run Script			All	

Script File	If Auto Run Script On	If Auto Run Script Off	Alpha	Limit of 14 characters

Controls in the GPI Application (continued)

Miscellaneous				
f-number	If using PSF or MTF	All others	Numeric	>= 0
Exit Pupil Diam	If using PSF or MTF	All others	Linear	>= 0
Refractive Index	If using Angles or PHOM application	All others	Numeric	>= 0
Part Thickness	If using PHOM app	All others	Linear	>= 0
Step Trim	If using Step Height app	All others	Linear	>= 0
Lot Num		All	Alpha	
Part Num		All	Alpha	
Part Ser Num		All	Alpha	
Measurement Id		All	Numeric	>=0
Comment		All	Alpha	
SC Pattern List	If using Pattern function	All others	Alpha	
CODE V Type	If exporting data to Code V	All others	List	Limited usefulness
Limits Error	All		Y/N	
Master Units				
Lateral Units	All			
Linear Units	All			
Height Units	All			
Angle Units	All			
Slope Units	All			
Volume Units	All			
Wavelength-Out	All			
Mouse				
Mouse Threshold	All			
Mouse Acceleration	All			

Controls in the Microscope Application

The following table is a list of control boxes found in the Control window in the basic Microscope application. They are listed to help you find a less-used or unused control that can be taken over when writing a script file. Refer to *Chapter 2* for more information on using input from Control boxes.

The Off Limits and Okay columns identify some of the instruments and/or conditions involved in using the control. "All" means that it is either off limits (or okay) for all instruments. "All others" means that other conditions are listed. The Units column identifies the units used with the control if you are looking for a particular type of control box, such as Numeric, alpha, or on/off.

Sub-menu/Control	Off Limits	Okay	Units	Comments
Interferometer				
Instrument	All			
Camera Mode	All			
Image Zoom	All			
Intf Scale Factor	All			

Wavelength-In	All			
Wavelength-In 1	2 wavelength microscopes	All others	Linear	0.1 - 100 microns
Wavelength-In 2	2 wavelength microscopes	All others	Linear	0.1 - 100 microns
Wavelength-In 3	2 wavelength microscopes	All others	Linear	0.1 - 100 microns
Wavelength-In 4	2 wavelength microscopes	All others	Linear	0.1 - 100 microns
Acquisition				
Acquisition Mode	All		List	
Scan Type	All		List	
Scan Length	SWLI microscopes	Not accessible	List	
Extended Scan Length	SWLI microscopes with extended range	All others	Linear	75 - 5308 microns
Wavelength Select	All		Alpha	
Two Wavelength	Two wavelength microscopes	All others	Y/N	
Wavelength Fold		Two wavelength microscopes	All others	Y/N
Intens Avgs		All		Numeric
AGC	All others	SWLI microscopes	Y/N	
AGC Mode	All others	"SWLI microscopes, not particularly useful"	List	
Light Level	All		Numeric	
Light Level Pct	All		Numeric	
LLC Delay	All others	SWLI microscopes	Numeric	0 - 10
Target Range	All others	SWLI microscopes	Numeric	0-100
Max Sat Pts	All others	SWLI microscopes	Numeric	>=0
Ignore Bright Pts	All others	SWLI microscopes	Y/N	
PZT Cal	All others	SWLI microscopes	Y/N	
PZT Gain	All others	SWLI microscopes and GPI in Intens Acquisition Mode	Numeric	0 - 4095
PZT Scale Factor	All others	SWLI microscopes and GPI in Intens Acquisition Mode	Numeric	0.125 - 2.0
Min Mod (%)	All			
Min Mod Pts	All			
Remove Fringes	All			
Phase Processing				
Fringe Threshold	GPI for fringe analysis	All others	Numeric	0 - 1
Fringe Limit	GPI for fringe analysis	All others	Numeric	0 - 10
Fringe Spacing	GPI for fringe analysis	All others	Numeric	0 - 1

Controls in the Microscope Application (continued)

Trim	All			
Trim Mode	All			
Phase Avgs	All			
Phase Avg Pause	All			
Phase Res	All			
FDA Res	SWLI microscopes	All others	List	Limited usefulness
Min Area Size	All			
Max Area Size	All			
Pre-Connect Filter	All			
Connection Type	All			
Connection Order	All			
Discon Action	All			
Discon Filter	All			
Remove Tilt Bias	All			
Remove	All			
Data Sign	All			
Phase Correction				
Phase Correction	All			
Test Region n	If Phase Correction On	If Phase Correction Off	Numeric	0.1 - 10
Test Region k	If Phase Correction On	If Phase Correction Off	Numeric	-10 - 10
Test Region Offset	If Phase Correction On	If Phase Correction Off	Height	
Ref Region n	If Phase Correction On	If Phase Correction Off	Numeric	0.1 - 10
Ref Region k	If Phase Correction On	If Phase Correction Off	Numeric	-10 - 10
Ref Region Offset	If Phase Correction On	If Phase Correction Off	Height	
Generate				
Surface Type	All others	Except when generating	List	Limited usefulness
Camera Size X	All others	Except when generating	Numeric	50 - 10500
Camera Size Y	All others	Except when generating	Numeric	50 - 10500
Camera Res	All others	Except when generating	Lateral	>= 0
Generate/Zernike				
Zernike Terms		Except when generating	List	Limited usefulness
Center X		Except when generating	Numeric	
Center Y		Except when generating	Numeric	
Radius		Except when generating	Numeric	
Coefs 0-15		Except when generating	Height	Scaled by Wavelength-In and Wavelength-Out, number will change if either of these values change
Coefs 16-35		Except when generating	Height	Same as Coefs 0-15
System Error				
Subtract Sys Err	All			
Sys Err File	If Subtract Sys Err On	If Subtract Sys Err Off	Alpha	Limit of 14 characters

Controls in the Microscope Application (continued)

Auto Sequence				
Alignment	If Adding/Subtracting data sets	All others	Y/N	
Alignment Tol	If using Alignment Fiducials	All others	Numeric	>= 0
Alignment Scaling	If using Alignment Fiducials	All others	List	Limited usefulness
Auto Seq Operation	If using Auto Sequence	All others	List	Limited usefulness
Auto Save Data	If using Auto Sequence	All others	Y/N	
Auto Save Data File	If Auto Save Data On	If Auto Save Data Off	Alpha	Limit of 14 characters
Auto Seq Max Count	If using Auto Sequence	All others	Numeric	> 0
Auto Seq Delay	If using Auto Sequence	All others	Numeric	> 0
Auto Seq Interval	If using Auto Sequence	All others	Numeric	> 0
Auto Focus				
Focus	All			
Focus Mode	If Focus On	All others	List	Limited usefulness
Focus Drop	If Focus On and using Phase Acquisition Mode	All others	Linear	0 - 25.4 microns
Focus Light Level Offset Pct	If Focus On	All others	Numeric	-50 - 50
Focus Min Mod (%)	If Focus On	All others	Numeric	1 - 100
Focus Min Contrast	If Focus On and using Phase Acquisition Mode	All others	Numeric	0 - 100
Focus Depth	If Focus On and using Phase Acquisition Mode	All others	Linear	1.27 - 508 microns
Focus Max Adjust	If Focus On	All others	Linear	0 - 2.54 mm
Focus Retry Max Adjust	If Focus On	All others	Linear	0 - 2.54 mm
Focus Offset	If Focus On	All others	Linear	-2.54 - 2.54 mm
Focus/Tilt X Offset	If Focus On	All others	Linear	-25.4 - 25.4 mm
Focus/Tilt Y Offset	If Focus On	All others	Linear	-25.4 - 25.4 mm
Focus/Tilt Delay	If Focus On	All others	Numeric	0 - 5
Auto Tilt				
Auto Tilt	All			
Auto Tilt Mode	If Auto Tilt On	All others	List	Limited usefulness
Auto Tilt Roll Offset	If Auto Tilt On	All others	Slope	-2 - 2 degrees
Auto Tilt Pitch Offset	If Auto Tilt On	All others	Slope	-2 - 2 degrees
Auto Tilt Roll Tol	If Auto Tilt On	All others	Slope	0 - 2 degrees
Auto Tilt Pitch Tol	If Auto Tilt On	All others	Slope	0 - 2 degrees
Auto Tilt Min Contrast	If Auto Tilt On	All others	Numeric	0 - 100
Auto Tilt Domain	If Auto Tilt On	All others	Slope	0.1 - 2 degrees
Auto Tilt Max Adjust	If Auto Tilt On	All others	Slope	0 - 4 degrees
Environmental				
Temperature (C)	If using IRS	All others	Numeric	-99 - 99
Pressure (Torr)	If using IRS	All others	Numeric	0 - 99999
Relative Humidity (%)	If using IRS	All others	Numeric	0 - 100
Index of Air		Attribute	Cannot	

be used

Controls in the Microscope Application (continued)

Masks				
Show Masks	All			
Auto Load Masks	All			
Masks File	If Auto Load Masks On	If Auto Load Masks Off	Alpha	Limit of 14 characters
Pattern				
Auto Load Pattern	All			
Pattern File	If Auto Load Pattern On	If Auto Load Pattern Off	Alpha	Limit of 14 characters
Pattern Number	If using pattern function	All others	Numeric	0 - 99
Stitch				
Type	If using stitching	All others	List	Limited usefulness
Display Measurements	If using stitching	All others	Y/N	
Auto Save Seq	If using stitching	All others	Y/N	
Seq File	If using stitching	All others	Alpha	Limit of 14 characters
Failure Action	If using stitching	All others	List	Limited usefulness
Auto Focus	If using stitching	All others	Y/N	
Store Process Stats	If using stitching	All others	Y/N	
Scanning Seq	If using stitching	All others	List	Limited usefulness
Stitch Processed Data	If using stitching	All others	Y/N	
Average Overlap Regions	If using stitching	All others	Y/N	
Diagnostics	If using stitching	All others	Y/N	
Auto Load Masks	If using stitching	All others	Y/N	
Masks File	If using stitching	All others	Alpha	Limit of 14 characters
N Cols	If using stitching	All others	Numeric	1 - 50
N Rows	If using stitching	All others	Numeric	1 - 50
Overlap (%)	If using stitching	All others	Numeric	10 - 90
Size X	If using stitching	All others	Linear	>= 0
Size Y	If using stitching	All others	Linear	>= 0
Zip Stitch				
Zip Distance	If using stitching	All others	Linear	>= 0
Auto Load Masks	If using stitching	All others	Y/N	
Masks File	If using stitching	All others	Alpha	Limit of 14 characters
Store Process Stats	If using stitching	All others	Y/N	
Lower Overlap (%)	If using stitching	All others	Numeric	10 - 90
Lower Size X	If using stitching	All others	Linear	>= 0
Lower Size Y	If using stitching	All others	Linear	>= 0
Use Lower Seq File	If using stitching	All others	Y/N	
Lower Seq File	If using stitching	All others	Alpha	Limit of 14 characters
Upper Overlap (%)	If using stitching	All others	Numeric	10 - 90
Upper Size X	If using stitching	All others	Linear	>= 0
Upper Size Y	If using stitching	All others	Linear	>= 0
Use Upper Seq File	If using stitching	All others	Y/N	
Upper Seq File	If using stitching	All others	Alpha	Limit of 14 characters
Upper Offset X	If using stitching	All others	Linear	
Upper Offset Y	If using stitching	All others	Linear	
Script				
Auto Run Script	All			

Script File	If Auto Run Script On	If Auto Run Script Off	Alpha	Limit of 14 characters

Controls in the Microscope Application (continued)

Miscellaneous				
f-number	If using PSF or MTF	All others	Numeric	>= 0
Exit Pupil Diam	If using PSF or MTF	All others	Linear	>= 0
Refractive Index	If using Angles or PHOM applications	All others	Numeric	>= 0
Part Thickness	If using PHOM applications	All others	Linear	>= 0
Step Trim	If using Step Height application	All others	Linear	>= 0
Lot Num		All	Alpha	
Part Num		All	Alpha	
Part Ser Num		All	Alpha	
Measurement Id		All	Numeric	>=0
Comment		All	Alpha	
SC Pattern List	If using Pattern function	All others	Alpha	
CODE V Type	If exporting data to Code V	All others	List	Limited usefulness
Limits Error	All		Y/N	
Master Units				
Lateral Units	All			
Linear Units	All			
Height Units	All			
Angle Units	All			
Slope Units	All			
Volume Units	All			
Wavelength-Out	All			
Mouse				
Mouse Threshold	All			
Mouse Acceleration	All			

Appendix D
Programmable Stage Pattern Functions

Patterns are used to control programmable stage hardware and to automate multiple measurements. In addition to the MetroPro operations used with programmable stages, there are many MetroScript functions and statements that can be used to manipulate patterns. They are listed below, and described in *Chapter 4* of this manual.

Function or Statement

contpattern
editpatpos
editpattern
getpatval(*StrExp1, Var, StrExpr2*)
•getsubpatval(*StrExp1, Var, StrExpr2*)
gotopatorg
gotopatpos(*IntExpr*)
loadpatpos(*FileNameExpr*)
loadpatstatus(*FileNameExpr*)
loadpattern(*FileNameExpr*)
•loadsubpat(*FileNameExpr*)
runpattern
savepatpos(*FileNameExpr*)
savepatstatus(*FileNameExpr*)
savepattern(*FileNameExpr*)
•savesubpat(*FileNameExpr*)
setpatorg
setpatpos(*Var1, Var2, Expr1, Expr2*)
setpatposstat(*IntExpr1, IntExpr2, StrExpr*)
setpatval(*StrExpr1, Expr, StrExpr2*)
•setsubpatval(*StrExpr1, Expr, StrExpr2*)
viewpatstatus

(• Indicates functions and/or statements that are associated with subpatterns.)

Subpatterns

Some of the important MetroScript functions and statements listed above are associated with a "sub" pattern. This is a subordinate pattern that is intended to be invoked at each position within a top-level pattern. Note that *scripting is required* to use the subpattern facilities; it is not possible to use the facilities via the MetroPro graphical user interface.

Within MetroScript, effectively hidden from view is a subpattern "holding area." There are statements for loading a pattern into the subpattern holding area and for accessing the subpattern values.

A typical use of the subpattern function is as follows. A subpattern is prepared using the Pattern Editor and saved to a file. Then a top-level pattern is prepared with the operation control set to Run Script. The Script Filename control specifies a custom script written by the user or an applications engineer. When the top-level pattern is run, the stage is moved to each top-level pattern position, and the script is invoked to perform the subpattern. The script must load the subpattern, move to each subpattern position and perform each measurement using functions such as **movexy**, **acquire**, **analyze**, **storeprocstats**, etc.

L

M

N

O

P

R

S

T

U

V

W

Z