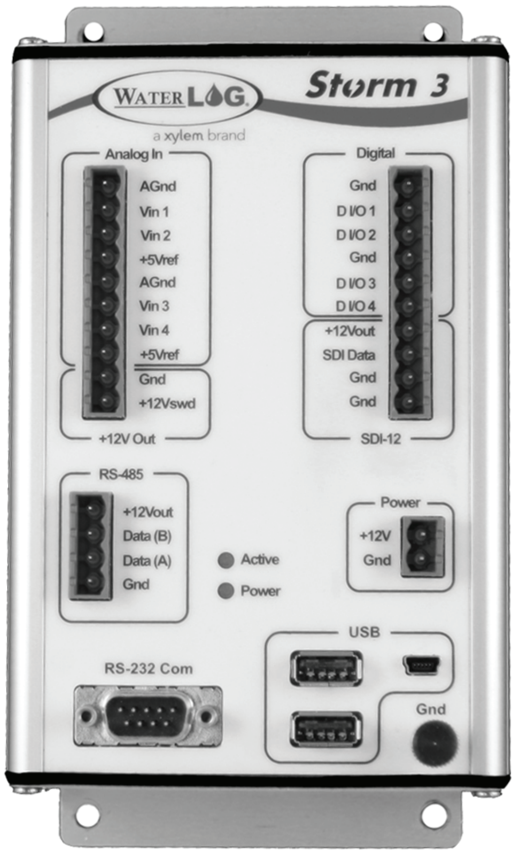
BASIC PROGRAMMING MANUAL

V1.0



Storm 3

D28 0214

CONTENTS

a xylem brand

Contents

01 / BASIC PROGRAMMING

Each Storm data logger contains a built-in BASIC interpreter (as of firmware version 1.4.2) allowing for more complex procedures and more specific control over operations performed by the data logger. Many traditional BASIC commands and features have been combined with an added subset of Storm-specific operations to create the Basic programming language available on the Storm. Programs can be easily created, edited, and debugged directly on the Storm, enabling quick and verified programs to enhance the Storm experience. This manual will cover all of the major Basic commands and provide examples on how they are used.

Basic Features

- User-defined number and string variables, including single and multi-dimensional arrays support
- Built-in string and number processing functions (for both received and user-created variables)
- Easy access to RS-232 COM/RS-485 serial ports using simple PRINT and INPUT commands
- Easy access to log and data files using simple PRINT and INPUT commands
- User-created subroutine support
- Support for traditional BASIC constructs such as goto, gosub, and line numbers
- Structured programming commands including SWITCH-CASE statements, single and multi-line IF-THEN statements, FOR, WHILE, REPEAT, and DO loops
- Ability to perform raw measurements of data logger Inputs (e.g. Analog, Digital, SDI-12, etc)
- Ability to retrieve current and previously measured values of any Sensor
- Access to system variables such as date and time
- Ability to perform complex math operations using built in trigonometric and logarithmic functions
- Ability to use a Basic program’s returned values within Outputs (Log File, GOES, etc)
- Ability to run a program in response to incoming RS-232 serial communication
- No limit to the number of variables created or used within a program
- No file size limit on Basic programs
- No limit to the number of Basic programs installed and/or used on the system
- Easy to install on or retrieve programs from the data logger
- Programs are saved as part of the configuration file
- Simple program troubleshooting via the User Interface debug option
- Ability to create and/or edit a Basic program directly on the User Interface
- Basic programs can be easily run on a scheduled basis or from a digital or serial event

BASIC PROGRAMMING

Basic Fundamentals

Understanding the available functions and commands as well as the overall operations and flow of Basic is essential to its use within the Storm. Primary Basic concepts and examples are provided below, followed by a listing of all available commands and functions. Each command is given a description and an example of its use. Additional examples are included at the end of this manual.

Before a Basic program can be used, it must first be installed on the Storm. This is done through the Sensors > Basic Programming screen. Basic programs should have a ".bas" extension to identify them as Basic programs. There is no limit to the number of programs that can be installed or used on the Storm

Scheduling a Program

Basic programs on the Storm can be set up to run on a timed schedule. A Basic program is set up similar to any other sensor through the Sensor > Add New Sensor screen. Selecting "Basic Program" from the available list will create the structure necessary to schedule a Basic program. When a program is selected, the program's internal variables are displayed as selectable items. Each variable's value can be captured and used after a program has run. These values can be used with one or more outputs, such as logging to a file, sending to GOES, or sending to Storm Central. Basic programs are evaluated according to their ordering within the sensor list, set by each sensor's Scan Order. Basic programs should be placed last in the Scan Order if the programs use values from other sensors measuring at the same scan time, in order to retrieve the latest measurement values.

Listening Programs

Basic programs can also be used to respond to "listen" and respond to communication on the RS-232 Com port. Programs can be used to respond to specific commands, make measurements, send data, etc. Using a listening program, data loggers or other serial devices (e.g. a PC or server) can communicate and send data back and forth. Listening programs are assigned under the Outputs > Communication Ports Setup screen in the Storm. Examples of listening programs can be found under the OPEN command.

Error Handling

Basic programs are parsed before execution and validatedthroughout. If any syntax errors are found in the code, the Basic program will abort and the Storm will continue its operations.

To assist with troubleshooting, the Sensors >Basic Programmingscreen contains a Debug section that allows line by line execution of an installed Basic program with detailed reporting of variables information and other related output. Any errors that are reported should be resolved before implementing the program into the Storm's schedule.

Grammar and Examples

Basic programming examples provided throughout the manual will follow a specific format. All Basiccommands and functions will be capitalized while all variables and strings may contain both uppercase and lowercase lettering. Though this format is used, the Basicinterpreter is not case-sensitive with respect to commands and functions. Variables, however, are case-sensitive. In other words, the following “PRINT”, “Print”, and “print” commands are all interpreted as valid PRINT commands, whereas variables “var” and “Var” are interpreted as two different variables. Code examples given in the manual will appear similar to the following:

```
PRINT “Hello World”
 var =12345.67
```

Comments

The REM statement (meaning remark) or single quote (‘) can be used to introduce a comment that extends to the end of the line. Comments can exist anywhere and span a single line. Once a comment has been started, the remainder of the line will also be considered a comment. Comments are not necessary in a program (they are ignored by the interpreter), though can provide helpful clarification when trying to understand. The following examples all represent valid uses of comments:

```
REM This is a comment
‘ This is also a comment
PRINT “Hello World”REM This is a valid comment after a command
PRINT “Hello World” ‘ This is also a valid comment after a command
```

Variables, Strings and Arrays

Unlike the case-insensitive nature of Basic statements, variables and strings are uniquely identified by their names. Thus a variable named “var” is understood to be different than a similarly named variable “Var”. Standard variables represent a number.

```
var = 15.2      REM sets var to 15.2
Var = 0REM sets Var 0
String variables are also available and require a trailing “$”.
var$ = “Hello World”  REM sets var$ to “Hello World”
```

Traditional variables do not need to be declared before their first use. Number variables are initialized to zero (“0”) while string variables are initialized to an empty string (“”). Arrays, on the other hand, need to be created first using the ARRAY command, prior to being used. Arrays contain multiple number or string variables. Array contents are initialized to zeros or empty strings depending on their declaration type.

```
REM create a single-dimension, two element number array
ARRAY myArray(2)
var$ = myArray(1), “ “, myArray(2)  REM sets var to “0 0”
```

Control Statements and Loops

Basic supports traditional control statements such as GOTO and GOSUB as well as single and multi-line IF-THEN statements and SWITCH-CASE statements. GOTO and GOSUB can jump to labels or line numbers within the code. Line numbers may be used in the Basic program, but they are not required.

Loops are also allowed using FOR, WHILE, REPEAT, and DO statements. BREAK statements may be used to leave any of the previously mentioned loops while the CONTINUE command can be used to begin the next iteration of a loop

Arithmetic, Comparison and Logical Operators

Traditional arithmetic, comparison, and logical operators are available for use within Basic and are noted below (ordered by precedence):

Arithmetic:	Comparison:	Logical:
^ (power)	= ==	NOT
- (unary minus)	<><= >=	AND
* / % (integer modulus)	<> !=	OR
+ -		

Math Functions

Basic provides a comprehensive list of math functions including logarithmic and trigonometric functions. All trigonometric functions use radians. The following functions are available: ABS, ACOS, ASIN, ATAN, CEIL, COS, EXP, FLOOR, FRAC, H377C, H377F, INT, LN, LOG, MAX, MIN, MOD, RND, SGN, SIN, SQR, SQRT, TAN, XOR. Constants EULERand PI are also available for use within Basic programs.

Subroutines

In order to ease the structure and flow of a Basic program, subroutines may be declared and used to perform various operations. The separation of these subroutines allows a more procedural approach to be taken within the code as well as simplify the reuse of segments of code in later programs. These subroutines can accept multiple parameters, numbers or strings, and can likewise return a number or a string, if desired. Subroutines are declared with the SUB command, ended with the END SUB command, and return values using the RETURN command.

```
REM Calculate the Volume of a Rectangle
height = 3
width = 4
depth = 5

volume = calc_volume(height, width, depth)  REM sets volume to 60

SUB calc_volume(h, w, d)
  vol = h * w * d
  RETURN vol
END SUB
```

File and Serial I/O

Files stored on the Storm's local file system can be accessed for reading, writing, and appending of data. The OPEN command allows files to be opened and assigned a number for use throughout the program. Using the INPUT and PRINT commands with the file's assigned number allows Basic to read and write to the opened file. Other commands such as EOF, LINE INPUT, SEEK, and TELL allow for further traversal and manipulation of data files.

The Storm's RS-232 Comand RS-485 ports can be opened, closed, read from, and written to with OPEN, CLOSE, INPUT, and PRINT commands as well.

More information and examples relating to both file and serial I/O can be found under each of the respective commands.

Escape Sequences and Hex Values

Special escape sequences are available for use within Basic. Non-printable characters such as a tab or newline can be created by a using a backslash ("\") followed by a single letter or hex value sequence as detailed below.

Additional printable and non-printable characters from the ASCII chart can be accessed by specifying a hexadecimal value, using the escape sequence \xXX, where XX is the hexadecimal value for the desired symbol in the ASCII chart. For example, from the table above, the carriage return character (\r) would be represented by the hexadecimal escape sequence \x0D.

Escape Sequences	ASCII Value	Hex Value	Description
\b	8	08	backspace
\t	9	09	horizontal tab
\n	10	0A	newline
\v	11	0B	vertical tab
\f	12	0C	form feed
\r	13	0D	carriage return
\"	34	22	double quote
\'	39	27	single quote
\\	92	5C	backslash
\xXX	N/A	N/A	hex value

String Processing

Basic provides a number of commands that allow for string manipulation and processing. Functions that return string variables contain a trailing "$" whereas functions returning number variables do not have a "$". Functions such as LEFT$, MID$, and RIGHT$ can be used for extracting parts of a string. Separating a string into usable sections can be done with TOKEN and SPLIT. Converting strings to numbers and numbers to strings can be quickly accomplished with the STR$ and VAL functions. The "+" operator may also be used to concatenate strings. There are many other string functions available including ASC, CHR$, HEX$, INSTR, LEN, LOWER$, MID$, UPPER$ that may be used to manipulate strings and their contents.

02 / COMMANDS AND FUNCTIONS

The Storm's implementation of Basic uses both commands and functions to perform actions and return values, though each is syntactically different. Commands such as INPUT and GETVALUE return values to the variable specified at the end of line (following a comma):

```
REM the following command stores the value of Analog 1 in variable a1
GETVALUEANALOG1, a1
```

Functions, on the other hand, specify the variable first, followed by the equal operator ("=") to assign the calculated value:

```
REM the following function stores the result in variable dt
dt = DATETIME(TIME)
```

Commands and Functions available to the Basic language are listed alphabetically below.

ABS (number)

Returns the absolute value of the given number.

```
var = ABS(-2.5)  REM sets var to 2.5
var = ABS(100)  REM sets var to 100
```

ACOS (number)

Returns the arc-cosine value of the given number.

```
var = ACOS(0)  REM sets var to 1.5708 (PI/2)
var = ACOS(0.5)  REM sets var to 1.0472 (PI/3)
```

AND

A logical operator used between two expressions in a conditional statement (e.g. IF, WHILE, etc.). Returns TRUE if the left and right expressions are TRUE, otherwise FALSE is returned.

```
var = 75
IF (var > 0 AND var < 50) var = 0REM does not set var (FALSE)
IF (var >50 AND var < 150) var = 100  REMsets var to 100 (TRUE)
```

ARRAY

Declares and creates an array. Arrays may be single or multi-dimensional and filled with numbers or strings. Number arrays, when created, are filled with zeros, while string arrays are filled with empty strings (""). To enlarge an array's size (not its dimension), re-declare the array with a larger size. Existing data will be preserved. Re-declaring an array with a smaller size does nothing. ARRAYDIM and ARRAYSIZE can be used to determine the number of dimensions of an array as well as the sizes of those respective dimensions.

Single-dimensional array:

```
ARRAY myArray(5)
FOR i = 1 TO 5
  myArray(i) = i * 2  REM Duplicate the number's value inside the array
NEXT i
REM myArray(i) now contains values 2 4 6 8 10
```

Multi-dimensional array:

```
ARRAY myArray$(3,7)  REM 2-dimensional string array (3 rows, 7 columns)
FOR i = 1 TO 3
  FOR j = 1 TO 7
    myArray$(i,j) = str$(i * j)  REM str$ converts a number to a string
  NEXT j
NEXT i
REM myArray$(i,j) now contains values 1 2 3 4  5  6  7
                                      2 4 6 8  10 12 14
                                      3 6 9 12 15 18 21
```

ARRAYDIM (array)

Returns the given array's number of dimensions.

```
ARRAY arr(10)
var = ARRAYDIM(arr()) REM sets var to 1 as arr is one-dimensional
ARRAY myArr(2,4,4)
var = ARRAYDIM(myArr()) REM sets var to 3 as myArr is three-dimensional
```

ARRAYSIZE (array, number)

Returns the given array's dimension's size, where array specifies the array and number specifies the dimension. The last parameter is optional and defaults to the first dimension if not specified.

```
ARRAY myArray(10)
var = ARRAYSIZE(myArray()) REM sets var to10
ARRAY myArr$(2,4)
var = ARRAYSIZE(myArr$(), 2) REM sets var to4
```

ASC (string)

Returns the first character's ASCII numerical representation. CHR$( ), converting a numerical value to the ASCII character, is the opposite function of ASC( ).

```
var = ASC("A")  REM sets var to 65
var$ = CHR$(65)  REM sets var$ toA
```

ASIN (number)

Returns the arc-sine value of the given number.

```
var = ASIN(0)  REM sets var to0
var = ASIN(1)  REM sets var to 1.5708 (PI/2)
```

ATAN (number)

Returns the arc-tangent value of the given number.

```
var = ATAN(0)  REM sets var to 0
var = ATAN(1)  REM sets var to 0.785398 (PI/4)
```

BREAK

Causes an immediate exit from a loop or SWITCH statement.

```
a = 0
WHILE (a < 10)
  a = a + 1
  IF (a > 5) BREAK  REM the while loop exits once a > 5
WEND
REM variable a is now 6
```

CASE

Used with the SWITCH statement to list potential routes.

```
a$ = "Hi"
SWITCH a$
  CASE "Hello":
      response$ = "And Hello to you"
      BREAK
  CASE "Hi":
      response$ = "Hi!"  REM this case matches and is used
    BREAK
END SWITCH
REM response$ is now "Hi!"
```

CEIL (number)

Returns the ceiling or smallest integer not less than the given number.

```
var = CEIL(1.2)  REM sets var to 2
var = CEIL(4.7)  REM sets var to5
```

CHR$ (number)

Returns the number's ASCII character representation. ASC( ), converting a character to its ASCII numerical value, is the opposite function of CHR$( ).

```
var$ = CHR$(65)  REM sets var to A
var = ASC("A")  REM sets var to 65
```

CLEARSDI ( )

Clears the SDI-12 cache. Prior to a Basic program running, the SDI-12 cache is empty. All Basic programs share the same SDI-12 cache. When an SDI-12 value is retrieved, a cache of the query and response is recorded. When a subsequent request is made, if the cache has the information available, the stored data is returned. To retrieve new measurements each scan, the cache should be cleared prior to retrieving fresh SDI-12 readings.

```
CLEARSDI()  REM Clear any cached SDI values
GETVALUE SDI01, a1$  REM request a new measurement (address 0, param 1)
GETVALUE SDI02, a2$  REM retrieve the second parameter (from the cache)
CLEARSDI()  REM Force our next request to be a new measurement
GETVALUE SDI01, b1$  REM request a new measurement (address 0, param 1)
```

CLOSE

Closes a file or serial port that has been previously opened with the OPEN command.

```
OPEN "RS-232 COM" AS #9
CLOSE #9

OPEN "RS-485" AS #6
CLOSE #6
```

CONTINUE

Used to begin the next iteration of a FOR, WHILE, REPEAT, or DO loop, skipping over the remainder of the loop.

```
sum = 0
FOR var = 1 TO 5
  IF (var = 3) CONTINUE  REM when var = 3, skip the next statement
sum = sum + var  REM sets sum to 12 (1 + 2 + 4 + 5)
NEXT var
```

COS (number)

Returns the cosine value of the given number.

```
var = COS(0)  REM sets var to 1
var = COS(PI)  REM sets var to -1
```

DATETIME

Used to retrieve parts of the current date and time. Available arguments are DATE, DATE$, TIME, TIME$, DAY, JDAY, MONTH, YEAR, SECONDS, MINUTES, HOURS, EPOCH, EPOCHDAY. DATE and TIME return integer values of the current date and time formatted as YYMMDD and HHMMSS respectively. The DATE$ and TIME$ arguments return a similar format as string values, specifically “MM/DD/YYYY” according to the Storm’s Date Format setting and “HH:MM:SS” respectively.

SECONDS, MINUTES, HOURS, DAY, MONTH, and YEAR each return a two-digit integer value of the specified current date or time subsection. JDAY returns a three-digit integer value of the Julian Day. EPOCH returns an integer value of the number of seconds since 00:00:00 Jan 1, 1970 while EPOCHDAY returns the number of seconds from 00:00:00 Jan 1, 1970 until the start of today’s date at 00:00:00.

```
REM Presuming today’s date and time are Sept 3, 2013 at 08:30:58
 var = DATETIME(DATE)          REM sets var to 130903
 var$ = DATETIME(TIME$)        REM sets var$ to 09/03/2013
 var = DATETIME(TIME)          REM sets var to 83058
 var$ = DATETIME(TIME$)        REM sets var$ to 08:30:58
 var = DATETIME(DAY)           REM sets var to 3
 var = DATETIME(JDAY)          REM sets var to 246
 var = DATETIME(MONTH)         REM sets var to 9
 var = DATETIME(YEAR)          REM sets var to 13
 var = DATETIME(SECONDS)       REM sets var to 58
 var = DATETIME(MINUTES)       REM sets var to 30
 var = DATETIME(HOURS)         REM sets var to 8
 var = DATETIME(EPOCH)         REM sets var to 1378197058
 var = DATETIME(EPOCHDAY)      REM sets var to 1378166400
```

DEFAULT

Used within a SWITCH-CASE statement to mark the default route if no other CASE statements are matched.

```
var = DATETIME(MINUTES)
path = 0
SWITCH var
  CASE 0:
    path = 1
BREAK
```

```
 CASE 15:
   path = 2
BREAK
 CASE 30:
   path = 3
BREAK
 CASE 45:
   path = 4
BREAK
 DEFAULT:
   path = -1
BREAK
END SWITCH
```

DELAY

Causes the program to pause execution for the specified number of seconds. A decimal number may be used to specify more specific and smaller time increments (e.g. milliseconds). SLEEP and DELAY are identical commands.

```
SLEEP 2.5       REM pauses the program for 2.5 seconds
DELAY 0.25      REM pauses the program for 0.25 seconds
```

DO

Begins an infinite loop encompassed by DO and LOOP. A BREAK or GOTO statement should be used to leave the loop.

```
DO
  REM Break out of the loop when our seconds are greater than 30
    x = DATETIME(SECONDS)
    IF (x > 30) BREAK
LOOP
```

ELSE

Optionally used as part of an IF statement to indicate a default branch if no other conditions are evaluated as TRUE.

```
var = DATETIME(MINUTES)
sync = 0
IF (var == 0) THEN
sync = 1
ELSE
sync = -1
ENDIF
```

ELSEIF

Optionally used as part of an IF statement to indicate another condition to evaluate. The option is evaluated if all previous options have resulted in FALSE. Multiple ELSEIF conditions may be tested If the condition evaluates to TRUE, all following conditions will not be tested.

```
var = DATETIME(MINUTES)
sync = 0
IF (var == 0) THEN
sync = 1
ELSEIF (var == 15) THEN
sync = 2
ELSEIF (var == 30) THEN
sync = 3
ELSEIF (var == 45) THEN
sync = 4
ELSE
sync = -1
ENDIF
```

END

Immediately ends the Basic program. Optional if used as the last statement.

```
var = DATETIME(MINUTES)
IF (var == 0) THEN
REM if top of the hour, end the program
```

ENDIF

Declares the end of a multi-line IF-THEN statement. Not required on single-line if statements.

```
var = DATETIME(MINUTES)
sync = 0
IF (var == 0) THEN
sync = 1
ELSEIF (var == 30) THEN
sync = 2
ELSE
sync = -1
ENDIF
```

END SUB

Declares the end of a subroutine (begun with the SUB statement).

```
volume = calc_volume(3, 4, 5)  REM sets volume to 60

SUB calc_volume(h, w, d)
  vol = h * w * d
  RETURN vol
END SUB
```

END WHILE

Marks the end of a conditional WHILE-WEND loop. END WHILE may also be used instead of WEND.

```
x = 0
WHILE (x < 30)
 REM Break out of the loop when our seconds are greater than 30
  x = DATETIME(SECONDS)
END WHILE
```

EOF (filenumber)

Returns TRUE if the given filenumber has reached the end of the file, FALSE if there is still data available to be read from the current position.

```
OPEN "SiteID.csv" FOR READING AS #1
WHILE (NOT EOF(#1))
  LINE INPUT #1, var$     REM reads each line of the log file
WEND
CLOSE #1
```

ERROR

Ends the Basic program with a custom error message. Primarily intended for troubleshooting Basic programs.

```
var = 12
IF (var > 10) THEN
  ERROR "Number too large!"          REM program ends here
ENDIF
```

EULER

A read-only constant containing the number 2.7182818284590452.

```
var = EULER  REM sets var to the euler constant
```

EXP (number)

Returnseuler raised to the power of the given number. Identical to EULER^number.

```
var = EXP(0)  REM sets var to 1, equivalent to EULER^0
var = EXP(1)  REM sets var to 2.71828, equivalent to EULER^1
var = EULER^2  REM sets var to 7.38906, equivalent to EXP(2)
```

FALSE

A read-only constant containing the number 0.

```
var = DATETIME(MINUTES)
top_of_hour = FALSE
IF (var ==0) THEN
  top_of_hour = TRUE
ENDIF
```

FLOOR

Returns the floor or largest integer not greater than the given number.

```
var = FLOOR(1.2)  REM sets var to 1
var = FLOOR(4.7)  REM sets var to 4
```

FLUSH

For files, writes any remaining unwritten (buffered) data to the specified file. For serial connections, removes any extra data on the given line. Closing a file or serial connection causes an automatic flush to occur.

```
OPEN “RS-232 COM” AS #2
INPUT #2, a$  REM retrieves from RS-232 COM and stores in string a$
PRINT #2 “a1”
FLUSH #2
INPUT #2, b$  REM retrieves from RS-232 COM and stores in string b$
CLOSE #2

val1 = 12.5
val2 = 32.6
OPEN “SiteID.csv” FOR WRITING AS #1
PRINT #1, “Appened values: “
FLUSH #1
PRINT #1, val1, “,”, val2
CLOSE #1
```

FOR

Begins a loop encompassed by FOR and NEXT with an optional STEP command. The default STEP is 1, meaning each iteration of the loop will increase the variable by 1. Negative STEPs may be used to count backwards.

```
FOR x = 1 TO 3
  REM retrieve the past 3 values of the Sensor named "Analog"
GETVALUE SENSOR "Analog" x, var$
NEXT x

FOR x = 6 TO 1 STEP -2
   REM retrieve every other past value of the Sensor named "Analog"
GETVALUE SENSOR "Analog" x, var$
NEXT x
```

FRAC (number)

Returns the fractional portion of the given number.

```
var = FRAC(26.245)  REM sets var to 0.245
```

GETVALUE

Used in conjunction with an identifier to request a new measurement or stored value on the Storm. Available identifiers are listed below. Astring or number variable must be specified after a comma to store the result. If an error is encountered or a value is not available, the Storm's Error Value (default is -99.99) is returned.

12VSWD

Returns the voltage reading of the switched +12 volt.See SetValue to turn on or off the +12Vswd reference.

```
GETVALUE 12VSWD, v12$    REM sets v12$ to the voltage of +12Vswd
```

5VREF

Returns the voltage reading of the +5V reference.See SetValue to turn on or off the +5V reference.

```
GETVALUE 5VREF, v5$          REM sets v5$ to the voltage on +5Vref
```

ANALOGX

Returns a new measurement from the specified Analog channel.TheXfollowing the identifier specifies which analog channelis measured.

```
GETVALUE ANALOG1, a1$   REM sets a1$ to the voltage on Analog channel 1
GETVALUE ANALOG2, a2$   REM sets a2$ to the voltage on Analog channel 2
```

COUNTERX

Returns a new counter measurement from the specified Digital port.TheXfollowing the identifier specifies which digital portis measured.See SetValue to change the counter value of a specific digital port.

```
GETVALUE COUNTER1, c1$ REM sets c1$ to the Digital port’s count
GETVALUE COUNTER2, c2$ REM sets c1$ to the Digital port’s count
```

DIGITALX

Returns a new measurement from the specified Digital port.A 1 or 0 will be returned based on the port’s level, 1 indicating the port is high (>3.5V), 0 indicating the port is low (<0.08V). TheXfollowing the identifier specifies which digital portis measured. See SetValue to change the level measurement of a specific digital port.

```
GETVALUE DIGITAL1, d1$     REM sets d1$ to 1 or 0, based on port level
GETVALUE DIGITAL2, d2$     REM sets d2$ to 1 or 0, based on port level
```

SDIXN or SDIXNN

Returns a specified parameter from a connected SDI-12 sensor. The X following the identifier specifies the SDI-12 sensor’s address. The N or NN denotes the parameter to return. For example, to request the 3rd parameter from an SDI-12 sensor on address 0, the identifier SDI03 would be used.

SDI-12 requests and responses are cached to allow quick retrieval of subsequent parameter requests. To force a new measurement to be made for a sensor, the CLEARSDI( ) function should be used to clear the SDI-12 cache. Prior to making SDI-12 requests in a Basic program, the CLEARSDI( ) function should probably be called to force new measurements.

```
CLEARSDI()
GETVALUE SDI01, sdi1$       REM sets sdi1$ to parameter 1 from sensor 0
GETVALUE SDI312, sdi2$      REM sets sdi12$ to parameter 12 from sensor 3
```

SENSOR

Returns a sensor’s most recent processed value. If an unknown sensor is specified, the Error String (default is -99.99) is returned. Prior processed values can be returned by adding an additional numerical parameter. The number 1 indicates the most recent measurement, 2 the previous measurement, 3 the value processed three scans prior, etc. Values from SDI-12 or Basic programs are retrieved by specifying the name of the sensor followed by the parameter’s name surrounded by parentheses.

```
GETVALUE SENSOR “H-377”, temp$
GETVALUE SENSOR “H-340SDI(Rainfall)”, stage$
GETVALUE SENSOR “H-340SDI(Rainfall)” 2, previous_stage$
```

SENSORRAW

Returns a sensor's most recent raw/unprocessed value. A raw value is the value of a measurement prior to slope, offset, function or digits are applied to it. If an unknown sensor is specified, the Error String (default is -99.99) is returned. Prior raw values can be returned by adding an additional numerical parameter. The number 1 indicates the most recent measurement, 2 the previous measurement, 3 the value processed three scans prior, etc. Values from SDI-12 or Basic programs are retrieved by specifying the name of the sensor followed by the parameter's name surrounded by parentheses.

```
GETVALUE SENSORRAW "H-377", temp$
GETVALUE SENSORRAW "H-340SDI(Rainfall)", stage$
GETVALUE SENSORRAW "H-340SDI(Rainfall)" 2, previous_stage$
```

SENSORFUNCTION

Returns a sensor's function, specified by the Use Function option under the Processing section of the Storm's Sensor Setup screen. If an unknown sensor is specified, an N/A is returned. Functions from SDI-12 or Basic programs are retrieved by specifying the name of the sensor followed by the parameter's name surrounded by parentheses. See SetValue to change the function of a sensor.

```
GETVALUE SENSORFUNCTION "H-377", temp$
GETVALUE SENSORFUNCTION "H-340SDI(Rainfall)", stage$
```

SYSBATT

Returns a new voltage measurement of the attached battery.

```
GETVALUE SYSBATT, b$      REM sets b$ to the voltage of +12V
```

SYSTEMP

Returns a temperature reading of the internal temperature sensor. The value is returned in degrees Celsius.

```
GETVALUE SYSTEMP, t$      REM sets t$ to the internal temperature
```

GOSUB

Branches to the specified LABEL or line number within the program. Once a RETURN statement is reached, execution is passed back to the statement immediately following GOSUB.

Subroutines, defined with the SUB command, provide a much more flexible method of executing and running portions of code and should be used instead of GOSUB.

```
h = 3
w = 4
d = 5
GOSUB 100  REM sets volume to 60
GOSUB ComputeArea  REM sets area to 12
END

LABEL ComputeArea
  area = h * w
RETURN

100
  volume = h * w * d
RETURN
```

GOTO

Jumps to the specified label or line number within the program. GOTO statements never return back to the point origin and thus have no RETURN statement. Subroutines, defined with the SUB command, cannot be exited with the GOTO statement.

```
h = 3
w = 4
d = 5
GOTO 100  REM sets volume to 60
GOSUB ComputeArea  REM is never run as above GOTO jumped below
END

LABEL ComputeArea
  area = h * w
RETURN

100
volume = h * w * d
```

H377C (number)

Returns the temperature, in degrees Celsius, of the given number (expecting a 0-5V analog voltage reading) based on the math equation for the WaterLog model H-377 temperature probe.

```
var = H377C(3.25)  REM sets var to 23.1779
```

H377F (number)

Returns the temperature, in degrees Fahrenheit, of the given number (expecting a 0-5V analog voltage reading) based on the math equation for the WaterLog model H-377 temperature probe.

```
var = H377F(3.25)  REM sets var to 73.7203
```

HEX$ (number)

Returns the hexadecimal string representation of the given number.

```
var$ = HEX$(32) REM sets var$ to 20
```

IF

Used to take actions based on the evaluation of given conditional statements. True is determined as anything non-zero; false is zero.

The short form of the IF statement does not include a THEN and must remain on one line. Multi-line IF statements contain the THEN keyword as well as ENDIF to mark the ending of the IF statement. IF statements may also contain the keywords ELSEIF to introduce alternative conditional statements as well as the ELSE keyword to provide a default path if no other conditions evaluate to true.

Single-line IF statement:

```
var = 250
IF (var > 200) var = 200  REM sets var to 200
```

Multi-line IF statement:

```
var = 40
IF (var >60) THEN
var = 60
ELSEIF (var >30) THEN
var = 30  REM sets var to 30
ELSE
var = 0
ENDIF
```

INPUT

Read from a file or serial port that has been previously opened with the OPEN command. An optional delimiter (stop character) may be specified for retrieving data. The default delimiter for files is a space; the end of transmission EOT (0x04) character is used for serial communication. When used with a serial port, an additional variable may be specified to set the timeout, in milliseconds, for reading from the port (default is 100ms). Note that files will not return the stop character whereas serial communication returns all characters (including the delimiter). Entire lines of files can be retrieved using the LINE INPUT command.

```
OPEN "RS-232 COM" AS #3
INPUT #3 250, a$  REM timeout is set to 250ms
INPUT #3, b$  REM reads again with default 100ms timeout
INPUT #3 "," 250, c$  REM read until a comma or 250ms passes
CLOSE #3

OPEN "RS-485" AS #2
INPUT #2 250, a$  REM timeout is set to 250ms
INPUT #2, b$  REM reads again with default 100ms timeout
INPUT #2 "," 250, c$  REM read until a comma or 250ms passes
CLOSE #2

OPEN "SiteID.csv" FOR READING AS #1
INPUT #1, d$  REM retrieves characters until a space is found
INPUT #1 "\t", e$  REM retrieves characters until a tab is found
INPUT #1 ",", f$  REM retrieves characters until a comma is found
```

INSTR (string, string, number)

Returns the position, starting at the position given by the third optional parameter (default is 1), of the first occurrence of the second given string within the first given string beginning at the left. If the string is not found, zero is returned. RINSTR may be used to begin searching for a string from the right, rather than the left. Zero is returned if the value is not found.

```
var = INSTR("A1,B2,C3,D4", "B2")  REM sets var to 4
var = INSTR("A1,B2,C3,D4,A1,B2", "B2", 5)  REM sets var to 16
```

INT (number)

Returns the integer portion of the given number.

```
var = INT(26.245)  REM sets var to 26
```

LABEL

Identifies a specific location by name. Commands such as GOTO and GOSUB can refer to and send execution to named LABELs. Line numbers are a special case of LABELs and do not require the LABEL prefix to be named.

```
h = 3
w = 4
d = 5
GOSUB 100  REM sets volume to 60
GOSUB ComputeArea  REM sets area to 12
END

LABEL ComputeArea
  area = h * w
RETURN

100
  volume = h * w * d
RETURN
```

LEFT$ (string, number)

Returns a string, starting from the left side, containing the given number of characters from the given string.

```
var$ = “A1,B2,C3,D4”
ab$ = LEFT$(var$, 5)REM sets ab$ to A1,B2
```

LEN (string)

Returns the length of the given string.

```
var = LEN(“A1,B2,C3,D4”)  REM sets var to 11
```

LINE INPUT

Reads an entire line from an open file.

```
OPEN “SiteID.csv” FOR READING AS #1
LINE INPUT #1, b$  REM retrieves the first line and stores it in b$
CLOSE #1
```

LN (number)

Returns the natural logarithm of the given number. The LOG function should be used if a common (base-10) logarithm is required.

```
var = LN(3)  REM sets var to 1.09861
```

LOCAL

Used within a subroutine, LOCAL marks the given variable as valid only within that subroutine. Variables within a subroutine are accessible anywhere within the program otherwise.

```
SUB change_var()
  LOCAL var
  var = 100
END SUB
var = 10
change_var()
REM var is still set to 10
```

LOG (number)

Returns the common (base-10) logarithm of the given number. The LN function should be used if a natural logarithm is required.

```
var = LOG(5)  REM sets var to 0.69897
var = LOG(10)  REM sets var to1
```

LOOP

Declares the end of a DO-LOOP statement.

```
DO
 REM Break out of the loop when our seconds are greater than 30
  x = DATETIME(SECONDS)
  IF (x > 30) BREAK
LOOP
```

LOWER$ (string)

Returns the given string as all lowercase.

```
var$ = LOWER$("A1,B2,C3,D4")  REM sets var$ to a1,b2,c3,d4
```

COMMANDS AND FUNCTIONS

LTRIM$ (string)

Returns the given string with all whitespace removed from only the left side.

```
var$ = LTRIM$("  A1,B2,C3,D4  ")  REM sets var$ to "A1,B2,C3,D4  "
```

MAX (number, number)

Returns the maximum value of the two given numbers.

```
a = RND()          REM generates a random number between 0 and 1
b = RND()          REM generates a random number between 0 and 1
c = MAX(a, b)      REM sets c to the largest of the two numbers
```

MID$ (string, number, number)

Returns a smaller section (substring) of a given string. The first parameter is the given string, the second is the starting point from the left side, and the third optional parameter dictates how many characters to return. If the third parameter is omitted, all remaining characters are returned.

```
var$ = MID$("A2: 12.5, VB: 12.250", 11)  REM sets var$ to VB: 12.250
myStr$ = "A1: 15, A2: 18, D3: 0"
var$ = MID$(myStr$, 9, 6)  REM sets var$ toA2: 18
```

MIN (number, number)

Returns the minimum value of the two given numbers.

```
a = RND()          REM generates a random number between 0 and 1
b = RND()          REM generates a random number between 0 and 1
c = MIN(a, b)      REM sets c to the smallest of the two numbers
```

MOD (number, number)

Returns the remainder of a division between the two given numbers. MOD performs floating-point division and returns the remainder. For integer-only division, use the % operator.

```
a = 102.5
b = 10
var = MOD(a, b)    REM sets var to 2.5 (the remainder of 102.5/10)
var = a % b        REM sets var to 2 (the remainder of 102/10)
```

NEXT

Declares the end of a FOR loop.

```
FOR x = 1 TO 10 STEP 2
 REM retrieve the every other value for the Sensor named "Analog"
GETVALUE SENSOR "Analog" x, var$
NEXT x
```

NOT

Negates the expression immediately following. The ! operator may alternatively be used.

```
OPEN "SiteID.csv" FOR READING AS #1
WHILE (NOT EOF(#1))
  LINE INPUT #1, var$ REM reads each line of the log file
WEND
CLOSE #1
```

ON number GOSUBlabels

Branches to one of a list of labels(with an expected RETURN) based on the given number. For example, if the given number is 1, the first GOSUB listed is used, if the number is 2, the second GOSUB, etc.

```
REM iterate through each label
height = 3
width = 4
depth = 5
FOR a = 1 TO 3
  ON a GOSUB calc_area,calc_volume,200
NEXT a
LABEL calc_area
area = height * width
RETURN

LABEL calc_volume
volume = area * depth
RETURN

200
END
```

ON number GOTO

Jumps to one of a list of labels(without an expected RETURN) based on the given number/argument. For example, if the given number is 1, the first GOTO listed is used, if the number is 2, the second GOTO, etc. Unlike GOSUB, GOTO statements never return back to the point of the GOTO.

```
GETVALUE SYSTEMP, temp
IF (temp > 40) THEN
   var = 1
ELSEIF (temp > 30) THEN
  var = 2
ELSE
  var = 3
```

```
ENDIF

ON var GOTO critical,moderate,okay

LABEL critical
SETVALUE DIGITAL1, 1
SETVALUE DIGITAL2, 1
SETVALUE DIGITAL3, 1
END

LABEL moderate
SETVALUE DIGITAL1, 0
SETVALUE DIGITAL2, 1
SETVALUE DIGITAL3, 1
END

LABEL okay
SETVALUE DIGITAL1, 0
SETVALUE DIGITAL2, 0
SETVALUE DIGITAL3, 1
END
```

OPEN

Opens a file, serial port, or connects to a listening port.All files and non-listening ports will be automatically closed when a program ends.

Files may be opened with one of three modes: READING, WRITING, or APPENDING. A file opened with WRITING mode will create a new file if the file doesn't exist, or erase the contents of an existing file. To prevent overwriting of an existing file, use APPENDING for write operations of an existing file. An alternativeOPEN syntax may be used to determine if a file exists, as shown below in the file example.

Serial ports are opened with the following defaults: baud rate = 9600, data bits = 8, parity = none, stop bits = 1, flow control = none, transmit char delay = 0, transmit line delay = 0. If other port settings are desired, they must be specified prior to opening the port using the SETPORT command, as shown below in the serial port example.

Listening ports are opened with settings specified in the interface under Outputs > Communication Ports Setup. The CLOSE command has no effect on a listening port. When communication occurs on a listening port, the assigned Basic program will be initiated. To send and receive data across the port, use the OPEN command as detailed in the example below. Subsequent INPUT and PRINT commands will read and write to the open port.

File:

```
OPEN “SiteID.csv” FOR APPENDING AS #1
PRINT #1, “A1,B2,C3,D4”
CLOSE #1

REM Check if file exists, OPEN #1 FOR READING will fail if it does not
FileExists = 1
FileName$ = “LogFile.csv”
IF (NOT OPEN(#1, FileName$)) THEN
  FileExists = 0
ELSE

  CLOSE #1
  OPEN FileName$ FOR READING AS #1
  LINE INPUT #1, line$
  CLOSE #1
ENDIF
```

Serial Ports (RS-232 and RS-485):

```
SETPORT 300, 8, none, 1, none, 50, 0  REM port settings
OPEN “RS-232 COM” AS #5  REM open at 300 baud w/50ms tx char delay
INPUT #5, var$
PRINT #5, “0M!”;  REM sends 0M!
CLOSE #5

SETPORT 115200, 8, none, 1, none, 00, 0  REM port settings
OPEN “RS-485” AS #4  REM open at 115200 baud w/0ms tx line or char delay
INPUT #4, var$
PRINT #5, “0M!”;  REM sends 0M!
CLOSE #4
```

Listening Port (RS-232 Com):

```
OPEN “LISTENER” AS #1  REM settings specified in Storm interface
PRINT #1 “Please enter a number: “
INPUT #1 5000, num$  REM wait for 5 seconds for input
PRINT #1 “\r\n”
PRINT #1 “You entered: “, num$, “\r\n”
CLOSE #1
```

OR

A logical operator used between two expressions in conditional statements (e.g. IF, WHILE, etc.). Returns TRUE if the left, right, or both expressions are TRUE, FALSE if both are FALSE.

```
var = 100
IF (var > 0 OR var < 50) var = 1       REM sets var to 1
IF (var > 0 OR var < 150) var = 1      REM sets var to 1
```

PI

A read-only constant containing the number 3.1415926535897932.

```
var = PI  REM sets var to 3.14159
```

PRINT

Writes characters to a file or serialport.An automatic carriage return and line feed (\r\n) are added to all PRINT statements used with file operations unless the statement is ended with a semicolon (;). PRINT statements used with serial operations send only the characters specified with no additional characters automatically appended. The USING statement may also be appended for added precision or to set the number of digits to print, specified with hashes "#".

Serial communication (RS-232 and RS-485):

```
OPEN "RS-232 COM" AS #4
REM Wake up the H-3531 sensor by sending a character
PRINT #4 "W"  REM Wakes up our sensor with a "W"
INPUT #4 6000, H3531$  REM retrieve our measurements (6 sec timeout)
CLOSE #4

OPEN "RS-485" AS #3
PRINT #3 "0316"  REM Send command to our sensor
INPUT #31000, var$  REM retrieve our measurements (6 sec timeout)
CLOSE #3
```

File communication:

```
OPEN "SiteID.csv" FOR WRITING AS #1
PRINT #1 "Digital2,AC-In"  REM \r\n automatically added
var = 1
var2 = 256.25
var3 = 12.565
PRINT #1 var, ",", var2, ",";  REM saves 1,256.25, \r\n not added
PRINT #1 var3 USING "##.##"  REM saves12.57 followed by \r\n
PRINT #1 "Done"  REM \r\n automatically added
CLOSE #1
```

REM

Begins a comment extending to the end of the line. An apostrophe (" ' ") may alternatively be used to start a comment.

```
REM This is a comment
' This is also a comment
var = 1  REM This is a valid comment
var = 2  ' This is also a valid comment
```

REPEAT

Begins a conditional loop encompassed by REPEAT and UNTIL. The condition is given after the UNTIL statement and while evaluated as TRUE, will continue to iterate through the loop.   Once the UNTIL condition is evaluated as FALSE, the loop exits.

```
REPEAT
 REM Break out of the loop when our seconds are greater than 30
  x = DATETIME(SECONDS)
UNTIL (x > 30)
```

RETURN

Returns control back to the calling line of a GOSUB call or a subroutine (SUB) call with an optional value.

```
REM Calculate the Area of a Rectangle
height = 3
width = 4

area = calc_area(height, width)  REM stores 12 in area

SUB calc_area(h, w)
a = h * w
  RETURN a
END SUB
```

RIGHT$ (string, number)

Returns a string, starting from the right side, containing the given number of characters from the given string.

```
var$ = "A1,B2,C3,D4"
cd$ = RIGHT$(var$, 5)REM sets cd$ to C3,D4
```

RINSTR (string, string, number)

Returns the position, starting at 1, of the first occurrence of the second given string within the first given string beginning at the right. If the string is not found, zero is returned. The third parameter is an optional number, defaulting to the last element of the string, indicating the starting position for the search. INSTR may be used to begin searching for a string from the left, rather than the right.

```
var = RINSTR("A1,B2,C3,D4,A1,B2", "B2")  REM sets var to 16
var = RINSTR("A1,B2,C3,D4,A1,B2", "B2", 5)  REM sets var to 4
```

RND (number)

Returns a random number. An optional number may be specified for the maximum range (from zero to, but not including, the given number). If no number is given, the default is 1.

```
x = RND()  REM sets x to a random number between 0 and 0.99999
y = RND(5)  REM sets y to a random number between 0 and 4.99999
```

RTRIM$ (string)

Returns the given string with all whitespace removed from only the right side.

```
var$ = RTRIM$("  A1,B2,C3,D4  ")  REM sets var$ to "  A1,B2,C3,D4"
```

SEEK

Sets the position from which the next INPUT statement will read from within an open file. The first parameter is the open file, the second parameter tells where to set the next read position, and the third is an optional setting of either "BEGINNING", "CURRENT", or "END". "BEGINNING" sets the count from the beginning of the file (and is the default if not specified), "CURRENT" counts from the current read position from within the file, and "END" counts from the end of the file. The TELL command retrieves the current read position of the given file.

```
REM Presuming SiteID.csv contains:
REM Digital1,Analog2,WindSpeed
REM 56.23,2.25,126.5

OPEN "SiteID.csv" FOR READING AS #1
SEEK #1, LEN("Digital1") + 1  REM sets our position to Analog2
INPUT #1 ",", var$  REM sets var$ to Analog2
SEEK #1, -11, "END"  REM sets our position to the Analog2 measurement
INPUT #1 ",", var2$REM sets var2$ to 2.25
CLOSE #1
```

SETPORT

Sets the serial port settings for unopened communication ports. Serial ports are opened with the following defaults: baud rate = 9600, data bits = 8, parity = none, stop bits = 1, flow control = none, transmit char delay = 0, transmit line delay = 0. If other port settings are desired, they must be specified prior to opening the port. Parity can be set to None, Even, or Odd. Flow Control can be set to None, Software, Hardware, or Both. Transmit delays are specified in milliseconds. Port settings only need to be set once and will affect all future port openings.

```
SETPORT 300, 8, none, 1, none, 50, 0  REM port settings
OPEN “COM 1” AS #5  REM open the COM at 300 baud w/50ms tx char delay
CLOSE #5
```

SETVALUE

Used in conjunction with an identifier to change settings on the Storm. Available identifiers are listed below.

12VSWD

Turns the +12Vswd referenceon or off.A 0 will turn it off, a value greater than zero will turn it on. See GetValue to retrieve +12Vswd voltage.

```
SETVALUE 12VSWD, 1          REM sets v12$ to the voltage of +12Vswd
```

5VREF

Turns the +5V referenceon or off. A 0 will turn it off, a value greater than zero will turn it on. See GetValue to retrieve +5Vref voltage.

```
SETVALUE 5VREF, 1   REM turns on the +5Vref
SETVALUE 5VREF, 0   REM turns off the +5Vref
```

COUNTERX

Sets a digital port counter value.

```
SETVALUE COUNTER1, 20    REM sets digital port 1 counter to 20
SETVALUE COUNTER2, 0     REM sets digital port 2 counter to 0
```

DIGITALX

Sets a digital port high (5V) or low (0V).A 0 set the port low, a value greater than zero will set it high.See GetValue to retrieve digital port’s status.

```
SETVALUE DIGITAL1, 1        REM sets digital port 1 high
SETVALUE DIGITAL2, 0        REM sets digital port 2 low
```

SENSORFUNCTION

Sets a sensor's function, specified by the Use Function option under the Processing section of the Storm's Sensor Setup screen. Setting a sensor's function will automatically enable the function for use as well. If an unknown sensor is specified, an N/A is returned. Functions from SDI-12 or Basic programs are retrieved by specifying the name of the sensor followed by the parameter's name surrounded by parentheses.

```
SETVALUE SENSORFUNCTION "H-377", "H377C(X)"
SETVALUE SENSORFUNCTION "H-340SDI(Rainfall)", "0"
```

SGN (number)

Returns -1, 0, or 1 indicating the sign (negative, zero, or positive respectively) of the given number.

```
var = SGN(-32.645)  REM sets var to -1
var = SGN(1023.98)  REM sets var to 1
```

SIN ( )

Returns the sine value of the given number.

```
var = SIN(PI/2)REM sets var to 1
var = SIN(0)     REM sets var to 0
```

SLEEP

Causes the program to pause execution for the specified number of seconds. A decimal number may be used to specify more specific and smaller time increments (e.g. milliseconds). SLEEP and DELAY are identical commands.

```
SLEEP 2.5       REM pauses the program for 2.5 seconds
DELAY 0.25      REM pauses the program for 0.25 seconds
```

SPLIT (string, array, string)

Splits apart a string into an array of strings. The first parameter provides the string to split apart. The second parameter provides the array which the new substrings will fill. The optional third parameter provides the delimiters or characters that determine where to split the primary string. If more than one delimiter is specified, splitting will occur at each character given. For example, if a comma and hyphen are given ",-", content will be split on any commas or hyphens found as in the example below. If no delimiter is specified, whitespace will be used as the delimiter (e.g. spaces and tabs). The number of strings found is returned.

Empty elements in the array are created if two delimiters are next to each other. The TOKEN function, on the other hand, does not add empty elements if two or more delimiters are in sequence.

The array will be automatically sized (larger or smaller) based on the number of strings that are produced by the split. The number of strings produced will also be returned by the function.

```
var$ = "A1,B2,C3,D4,-A1,-B2"
ARRAY arr$(1)
array_num = SPLIT(var$, arr$(), ",-")  REM split on commas, returns 8
REM array arr$ now contains [A1][B2][C3][D4][][A1][][B2]
```

SQR (number)

Returns the squared value of the given number. The caret ("^") may also be used to raise a number to a power, such as 2.

```
var = SQR(5)   REM sets var to 25
var = 4^2                REM sets var to 16
```

SQRT (number)

Returns the square root value of the given number. The caret ("^") may also be used to raise a number to a power, having the same effect by using a fraction such as 1/2.

```
var = SQRT(25)  REM stores 5 in the var variable
var = 16^(1/2)  REM sets var to 4
```

STEP

Specifies an optional increment in a FOR loop. Positive or negative numbers may be used.

```
FOR a = 10 TO 1 STEP -2
  var = a  REM sets var to  10 8 6 4 2
NEXT a
```

STR$ (number, string)

Returns the string equivalent of the given number. An optional second parameter string may be specified to declare the format to be used. Formats are specified using the hash character ("#"). If not enough hashes are specified, the format will be returned. See VAL( ) for converting a string to a number.

```
var$= STR$(18.265)              REM sets var$ to 18.265
var$ = STR$(18.265, "#.#####")  REM sets var$ to #.#####
var$ = STR$(18.265, "##.####")  REM sets var$ to18.2650
var$ = STR$(18.265, "#######")  REM sets var$ to     18
```

SUB

Declares a user-defined subroutine. Subroutines can specify and accept multiple parameters and can return a number or string value using the RETURN statement. The END SUB statement marks the end of a subroutine. Though not required, it is recommended that any subroutine returning a string value should name routine with a trailing dollar-sign ("$").

```
area$ = calc_area$(3, 4)             REM sets area$to 12
volume = calc_volume(3, 4, 5)        REM sets volume to 60

SUB calc_area$(h, w)
  area = h * w
  RETURN STR$(area)
END SUB

SUB calc_volume(h, w, d)
  vol = h * w * d
  RETURN vol
END SUB
```

SWITCH

Declares the beginning of a SWITCH-CASE clause. The single variable given after the SWITCH keyword specifies the character or characters to match. Once a CASE match is made, the program will continue until a BREAK statement is seen, at which point the program will continue execution at the END SWITCH statement. An optional DEFAULT route is followed if no other CASE statements cause a match to occur.

```
var = 1
SWITCH var
  CASE 0:
response$ = "Too low"
    BREAK
  CASE 1:
  CASE 2:
    response$ = "Low"
BREAK
  CASE 3:
  CASE 4:
    response$ ="Mid"
BREAK
  CASE 5:
  CASE 6:
response$ = "High"
    BREAK
  DEFAULT:
    response$ ="Unknown number"
END SWITCH
```

TAN (number)

Returns the tangent value of the given number.

```
var = TAN(PI/4)        REM sets var to 1
var = TAN(0)           REM sets var to 0
```

TELL (filenumber)

Returns the current read position of the given file number.

```
REM Presuming SiteID.csv contains:
REM Digital1,Analog2,WindSpeed
REM 56.23,2.25,126.5

OPEN "SiteID.csv"      FOR READING AS #1
INPUT #1, var$         REM sets var$ to Digital1,Analog2,WindSpeed
var = TELL(#1)         REM sets var to 27
SEEK #1, -6, "CURRENT"
var = TELL(#1)         REM sets var to 21
INPUT #1, var$         REM sets var$ to Speed
CLOSE #1
```

THEN

Used with the IF statement to specify a multi-line IF-THEN clause.

```
var = 40
IF (var >60) THEN
var = 60
ELSEIF (var >30) THEN
var = 30  REM sets var to 30
ELSE
var = 0
ENDIF
```

TO

Used with the FOR statement to specify the range of the loop.

```
FOR x = 1 TO 3
  REM retrieve the past 3 values of the Sensor named "Analog"
GETVALUE SENSOR "Analog" x, var$
NEXT x
```

TOKEN (string, array, string)

Splits apart a string into an array of strings. The first parameter provides the string to split apart. The second parameter provides the array which the new substrings will fill. The optional third parameter provides the delimiters or characters that determine where to split the primary string. If more than one delimiter is specified, splitting will occur at each character given. For example, if a comma and hyphen are given ",-", content will be split on any commas or hyphens found as in the example below. If no delimiter is specified, whitespace will be used as the delimiter (e.g. spaces and tabs).The number of strings found is returned.

Empty elements in the array are not created if two delimiters are next to each other. The SPLIT function, on the other hand, does add empty elements if two or more delimiters are in sequence.

The array will be automatically sized (larger or smaller) based on the number of strings that are produced by the split. The number of strings produced will also be returned by the function.

```
var$ = "A1,B2,C3,D4,-A1,-B2"
ARRAY arr$(1)
array_num = TOKEN(var$, arr$(), ",-")  REM split on commas, returns 6
REM array arr$ now contains [A1][B2][C3][D4][A1][B2]
```

TRIM$ (string)

Returns the given string with all whitespace removed from both the left and right side.

```
var$ = TRIM$("  A1,B2,C3,D4  ")  REM sets var$ to A1,B2,C3,D4
```

TRUE

A read-only constant containing the number 1.

```
var = DATETIME(MINUTES)
top_of_hour = FALSE
IF (var ==0) THEN
  top_of_hour = TRUE
ENDIF
```

UNTIL

Marks the end of a conditional loop encompassed by REPEAT and UNTIL. The condition is given after the UNTIL statement and while evaluated as TRUE, will continue to iterate through the loop. Once the UNTIL condition is evaluated as FALSE, the loop exits.

```
REPEAT
 REM Break out of the loop when our seconds are greater than 30
  x = DATETIME(SECONDS)
UNTIL (x > 30)
```

UPPER$ (string)

Returns the given string as all uppercase.

```
var$ =UPPER$("a1,b2,c3,d4")  REM sets var$ to A1,B2,C3,D4
```

USING

Used with the PRINT statement to specify the format of the given number. Format is specified using hashes ("#"). If more hashes are provided than needed, the beginning is space-filledand the end zero-padded. If not enough hashes are given before the decimal, the hash format is returned.

```
var = 128.265
OPEN "SiteID.csv" FOR WRITING AS #2
PRINT #2, var USING "####.###"      REM prints  128.265
PRINT #2, var USING "###.####"      REM prints 128.2650
PRINT #2, var USING "###.##"                REM prints 128.26
PRINT #2, var USING "##.##"         REM prints ##.##
PRINT #2, var USING "###"           REM prints 128
CLOSE #2
```

VAL (string)

Returns a number equivalent for the given string. Whitespace is ignored. See STR$( ) for converting a number to a string.

```
var = VAL("26.250")   REM sets var to 26.250
var = VAL("  22  ")   REM sets var to 22
```

WEND

Marks the end of a conditional WHILE-WEND loop. END WHILE may also be used instead of WEND.

```
x = 0
WHILE (x < 30)
 REM Break out of the loop when our seconds are greater than 30
  x = DATETIME(SECONDS)
WEND
```

WHILE

Marks the beginning of a conditional WHILE-WEND loop. The condition is specified after the WHILE keyword. As long as the condition evaluates to TRUE, the loop will continue to iterate. Once the condition evaluates to FALSE, the loop will exit.

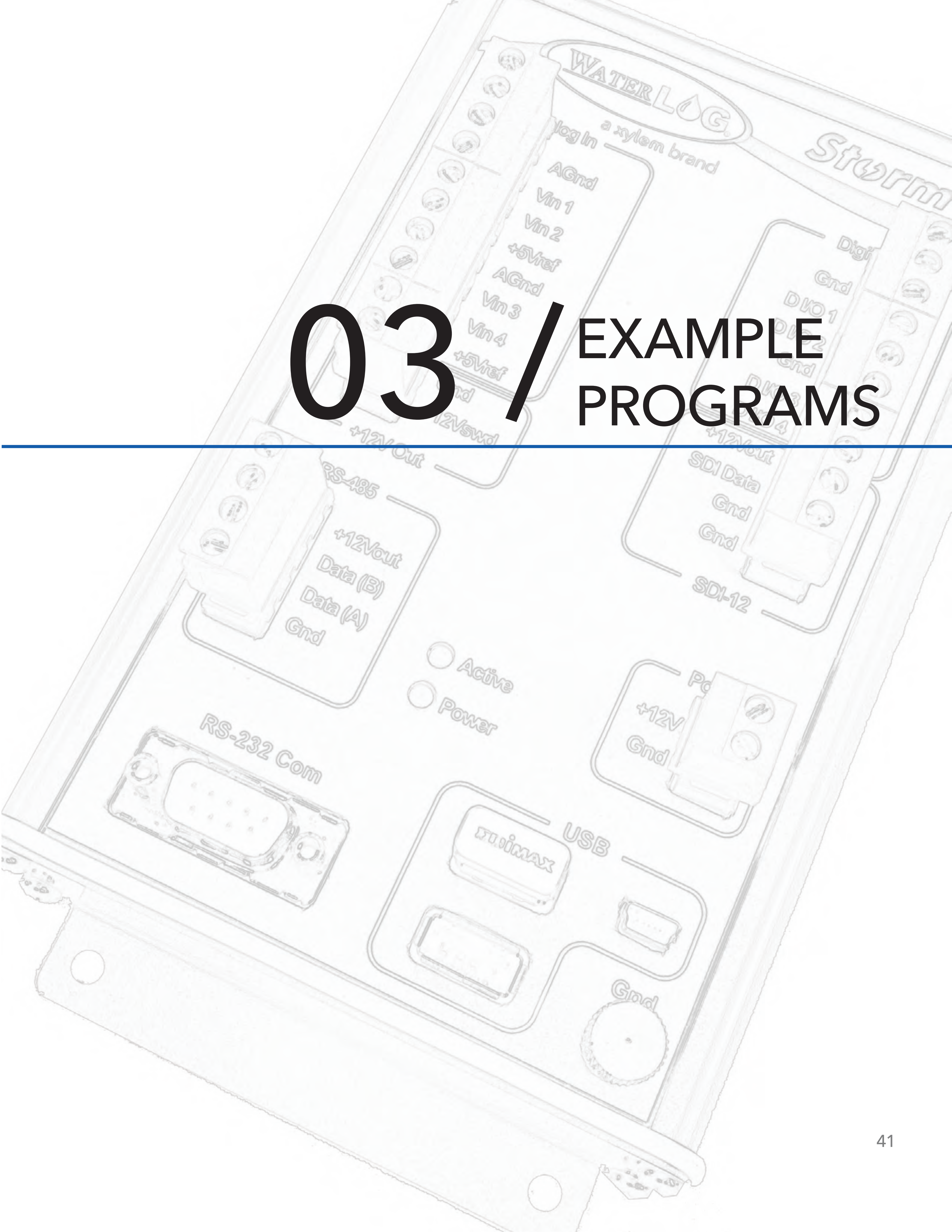
```
REM Presuming SiteID.csv contains:
REM Digital1,Analog2,WindSpeed
REM 56.23,2.25,126.5

OPEN "SiteID.csv" FOR READING AS #1
WHILE (!EOF(#1))
  INPUT #1 ",", var$      REM reads each name and value from the file
WEND
CLOSE #1
```

XOR (number, number)

Returns the bitwise exclusive or (as a number) of the two numeric parameters.

```
var = XOR(6, 2)         REM sets var to XOR: 4
```

03 / EXAMPLE PROGRAMS



EXAMPLE PROGRAMS

The following programs are provided for reference when creating and using the Basic interpreter.

AutoPrint.bas

```
REM    Prints out the last line of the log file to the RS-232 Com port
REM    When set up to run with the sensors, prints out their values every scan

LogFile$ = “SiteID.csv”

OPEN LogFile$ FOR READING AS #1
SEEK #1, 0, “END”
fileSize = TELL(#1)
filePosition = TELL(#1)

fileError = 0
foundEnd = 0
IF (filePosition < 150) THEN
        REM The first search should begin at the start
        SEEK #1, 0, “BEGINNING”
ELSE
        REM Note that this is presuming that each log line is less than 150 characters long
        SEEK #1, -150, “END”
END IF

WHILE (foundEnd == 0 and fileError == 0)
        LINE INPUT #1, LastLogLine$
        filePosition = TELL(#1)
        IF (filePosition == fileSize) THEN
                foundEnd = 1
        END IF
WEND
CLOSE #1

SETPORT 9600, 8, none, 1, none, 50, 0
OPEN “RS-232 Com” AS #2
PRINT #2 LastLogLine$, “\r\n”
CLOSE #2
```

CompassCorrection.bas

```
REM Determines true wind direction as a bouy rotates,
REM     based off recorded compass and wind direction

GETVALUE SENSOR “Compass”, WindDirVal
GETVALUE SENSOR “WD”, CompassVal

TrueWindDirVal = WindDirVal + CompassVal

IF (TrueWindDirVal >= 360) THEN
        TrueWindDirVal = TrueWindDirVal - 360
END IF
```

SensorAvg.bas

```
REM Calculates the running average for a specific sensor

SensorName$ = “SystemTemperature”
Measurements = 4
CombinedValue = 0

FOR n = 1 TO Measurements
        GETVALUE SENSOR SensorName$ n, SensorValue
        CombinedValue = CombinedValue + SensorValue
NEXT n

AverageValue = CombinedValue / Measurements
```

SensorMax.bas

```
REM Finds the maximum measurement of the given sensor’s last n values

SensorName$ = “TempC”
Measurements = 4
MaximumValue = -99999

FOR n = 1 TO Measurements
        GETVALUE SensorName$ n, SensorValue
        IF (SensorValue > MaximumValue) THEN
          MaximumValue = SensorValue
        END IF
NEXT n
```

EXAMPLE PROGRAMS

Listener.bas

```
REM Respond to simple commands on the RS-232 Com port as a Listener program

OPEN “LISTENER” AS #3
PRINT #3 “Enter Command > “

50
REM Ten seconds of inactivity causes the program to exit
INPUT #3 “\n” 10000, reply$

IF (reply$ == “”) THEN
        END
ELSE
        REM Trim trailing carriage return or line feed
        DO
           last_char$ = RIGHT$(reply$, 1)
              IF (last_char$ == “\n”) OR (last_char$ == “\r”) THEN
                  reply$ = LEFT$(reply$, LEN(reply$) - 1)
                ELSE
                  BREAK
                END IF
        LOOP
ENDIF

REM remove case sensitivity from command
reply$ = UPPER$(reply$)

SWITCH reply$
        CASE “BATTERY?”:
           GETVALUE SYSBATT, var
           PRINT #3 “Battery = “, var USING “##.#”, “\r\n”
        BREAK

        CASE “LASTSTAGE?”:
           GETVALUE SENSOR “Stage”, var
           PRINT #3 “Stage = “, var, “\r\n”
        BREAK

        CASE “ANALOG1?”:
           GETVALUE ANALOG1, var
           PRINT #3 “Analog 1 = “, var USING “##.###”, “\r\n”
        BREAK

        DEFAULT:
           PRINT #3 “Unknown command \””, reply$,”\” \r\n”
        BREAK
END SWITCH
GOTO 50
```

VR2C.bas

```
REM Communicate with and retrieve data from an RS-232 sensor

SerialNum$ = "450065"

SETPORT 9600,8,None,1,None,0,0
OPEN "RS-232 COM" AS #4
PRINT #4 "\r"     REM Wake up sensor
SLEEP 0.100       REM Wait for sensor to fully wake up

REM COMMAND FORMAT: *SSSSSS.P#CC,command\r
REM     SSSSSS = serial number of device (passed in)
REM     P = 0
REM     CC = decimal summation of SSSSSS and P
SerialNumLength = LEN(SerialNum$)
SerialNumSum = 0
SerialNumChar = 1
WHILE (SerialNumLength > 0)
        SerialNumSum = SerialNumSum + VAL(MID$(SerialNum$, SerialNumChar, 1))
        SerialNumChar = SerialNumChar + 1
        SerialNumLength = SerialNumLength - 1
WEND
IF (SerialNumSum < 10) THEN
        SerialNumSum$ = "0" + STR$(SerialNumSum)
ELSE
        SerialNumSum$ = STR$(SerialNumSum)
END IF
PRINT #4 "*", SerialNum$, ".0#", SerialNumSum$, ",RTMNOW\r"
INPUT #4 "\r" 5000, ack$         REM retrieve the acknowledgement
INPUT #4 "\r" 5000, status$      REM retrieve the status line
INPUT #4 ">" 20000, reply$
CLOSE #4

REM Remove initial <CR><LF> from reply
NLPosition = INSTR(reply$, "\n")
IF (NLPosition < 3) THEN
        reply$ = MID$(reply$, NLPosition + 1)
END IF

REM Remove the trailing greater than symbol and <CR><LF> from reply
GTPosition = INSTR(reply$, ">")
IF (GTPosition > 0) THEN
        reply$ = MID$(reply$, 1, GTPosition - 3)
END IF

IF (LEN(reply$) > 5) THEN
        OPEN "VR2CData.csv" FOR APPENDING AS #3
        PRINT #3, reply$
        CLOSE #3
END IF
```

Xylem

1) The tissue in plants that brings water upward from the roots;
2) a leading global water technology company.

We're 12,000 people unified in a common purpose: creating innovative solutions to meet our world's water needs. Developing new technologies that will improve the way water is used, conserved, and re-used in the future is central to our work. We move, treat, analyze, and return water to the environment, and we help people use water efficiently, in their homes, buildings, factories and farms. In more than 150 countries, we have strong, long-standing relationships with customers who know us for our powerful combination of leading product brands and applications expertise, backed by a legacy of innovation.

For more information on how Xylem can help you, go to www.xyleminc.com

xylem
Let's Solve Water

WATERLOG
a xylem brand

Xylem—WaterLOG
95 W 100 S, Suite 150
Logan, UT 84321
Tel +1.435.753.2212
Fax +1.435.753.7669
www.waterlog.com