Hurricane Macro (HMac)

OPERATOR GUIDE

February 2014

Cansec Systems Ltd.

Cutting edge simplicity

Table of Contents

Conventions

This document uses the following conventions:

Macro Keywords

Keywords recognized by the macro compiler will be displayed in blue type as in the following example: Repeat

Parameters and Other Macro Arguments

Parameters and other information associated with commands are displayed in italics.

Source Code Examples

Source code examples will be displayed in the following fixed font:

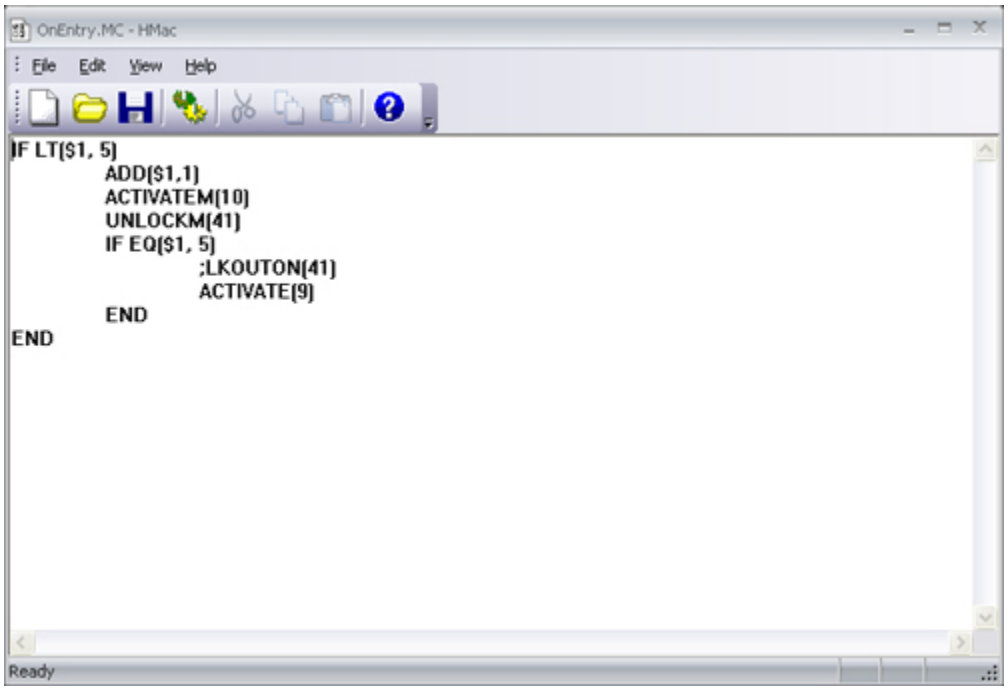
```
Repeat(10)
        Unlock(1)
End
```

Introduction

Overview

A macro program is a set of instructions that can be executed by Hurricane. They can be used to help automate certain tasks performed by an operator such as unlocking a group of doors, or more sophisticated logic requiring decisions based on device status and executed automatically in response to a system event. In this section we will introduce the Hurricane HMac macro compiler and explore the basic structure and steps required for creating a Hurricane Macro. We will also review the different ways that a macro can be executed from within the Hurricane system.

HMac Editor and Compiler



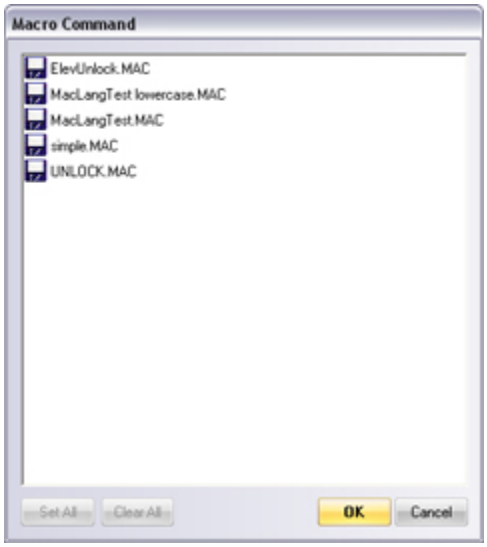
The HMac Editor and Compiler is available from the Start menu or from Hurricane's Tools menu. The program provides a basic text editor interface for inputting macro commands and statements. Commands for opening and saving macro files are available from the File Menu.

A macro consists of two separate files, a text file known as the source file ending in the extension .mc and a file executable by Hurricane ending in the extension .mac. The source file contains a sequence of lines called statements. Typically, these statements will be commands to control devices, or check the state of a system device. This source file is submitted to a process known as compilation. Compilation is the means by which the macro statements are converted into a format understandable by Hurricane. The compilation process results in a new file by the same name, but with the extension .mac being created. It is these final .mac files which will be specified from within Hurricane.

Running Macros in Hurricane

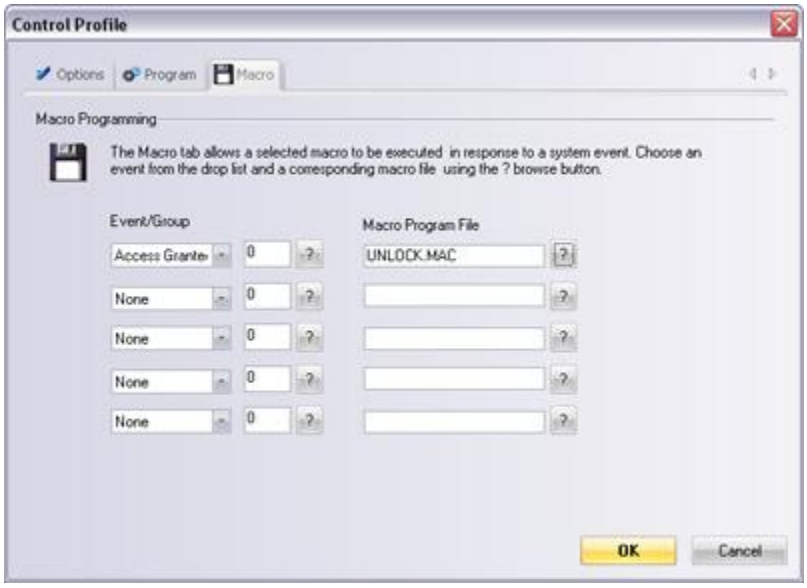
Macros can be executed in several different ways from within the Hurricane system. This topic provides a brief outline of these methods.

Manually by the Operator



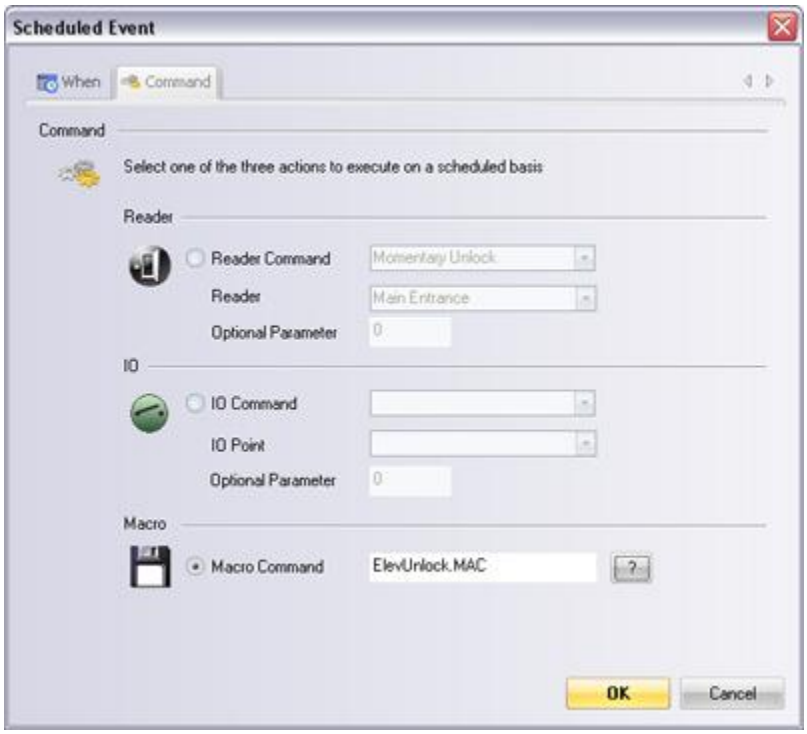
A Macro can be selected for execution by an operator by choosing the **Commands -> Run Macro** command. A selection list of available macros will presented from which one can be chosen for execution.

By a Control Profile



Hurricane Control Profiles allow macros to be executed in response to system events. A single Control Profile has provision for 5 unique event/macro combinations.

As a Scheduled Event



Hurricane allows device commands and macros to be scheduled to execute on a one time or recurring basis. Select the type of scheduling and the macro option when defining a new event.

Within Graphic Maps

Hurricane graphic maps support programmable buttons which can execute macros when clicked by the operator.

Runtime Variables

Macros can take advantage of information available at the time the macro is executed. If a macro is executed in response to a control profile, the logical id number of the device which triggered execution is stored in the runtime variable DevId. If the macro executes in response to a reader event, DevId will contain the reader's logical id number (1 through 512). Alternatively, if an I/O point triggers the macro, the point's logical id number will be present (1 through 4096). The DevId keyword can be used anywhere a macro command accepts a device id number. It can also be used in counter test expressions. The DevId variable allows the macro to test for specific readers or IO points and perform different actions accordingly.

Also available is the runtime variable CardId. If the Control Profile event has a cardholder id number associated with it, this value will be set into this variable for subsequent expression testing purposes by the macro.

In the case where a macro is not executed via a control profile, both of these variables will be set to 0.

A Simple Example

The Hurricane macro language consists of a set of predefined keywords with optional parameters. All macro keywords are case insensitive and may be entered in lower, upper or a mix of cases. The macro compiler also recognizes the symbol **:** as the start of a comment which extends to the end of the current line. All characters after the **:** are ignored and allows the user to enter comments or other descriptive text documenting the purpose and/or meaning of the macro file.

In the remainder of this topic we will consider the steps required to create, build, and run a macro file using an example. Our first step is to create a source file, SIMPLE.MC containing three statements:

```
Unlock(1)
Wait(10)
Relock(1)
```

The first statement Unlock(1), unlocks the reader defined with logical reader number 1. The second statement Wait(10), tells Hurricane to wait or pause for 10 seconds. The third statement Relock(1) re-locks reader 1. The result of running this macro in response to a device change of state would be the same as executing a momentary unlock command.

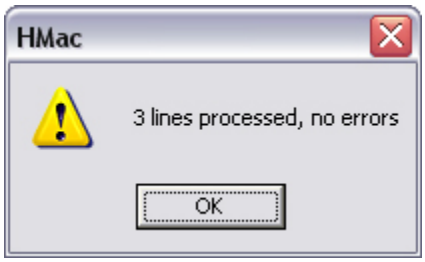
Step 1

We begin by entering the commands as listed above. When you start HMac, you will notice the three different areas: the tool and menu bars, and the edit portion of the window. In the edit area the cursor will be flashing at the upper left corner. Begin by typing in the commands as shown above. The source file must be saved and given a name, SIMPLE.MC for this example. Use the **File -> Save As** command from the menu bar. You will you be prompted for a filename, and here you may enter Simple. HMac will automatically add the file extension .MC to your filenames. We have now created our first macro source file.

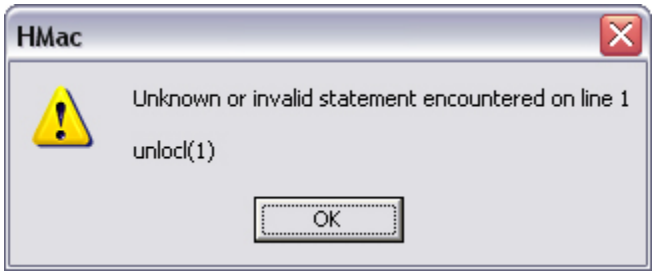
Step 2

The next step is to build our macro file. Use the **File -> Compile** command to translate the source file into the macro SIMPLE.MAC. It is this name that would be entered as a macro filename under a Control profile's Macro tab or a Scheduled Event's Macro section.

If the build process is successful, HMac will pop up a message box saying,



and you will know that you have typed the sample correctly. If you have made an error when entering this sample, say you mistakenly typed in, Unlocl(1) for the first line, HMac will pop up,



These errors will not produce a macro command file and must be corrected. When you receive the No Errors pop up, you can continue on to the next step.

Step 3

We are now in a position to run the Simple.MAC macro file. From the Hurricane client software, select the **Commands -> Run Macro** command from the menu. You should see your Simple.MAC file in the selection box. Select it and press **OK** to run this macro. You should observe reader 1 unlocking and after a delay of 10 seconds, reader 1 re-locking.

Device Commands

Device Commands

One of the primary reasons for creating a macro program is to control or command a device in some manner. For example, one may need to unlock a door or shunt an input point under some circumstances. The macro device commands provide the means to do this.

Three basic field devices are supported: the card reader, input point, and output point. In addition, commands are available to print messages on printers, and even transmit data out RS232 serial ports. We will begin by looking at the field device commands first.

Five reader commands are available: Relock, Unlock, Unlockm, Lkouton, Lkoutoff. Relock and Unlock are self-explanatory. Unlockm corresponds to the Hurricane Unlock Momentary command. Lkouton and Lkoutoff enable and disable lockout mode on a reader. A logical reader number as defined in Hurricane or the runtime DevId keyword follows each command. These reader numbers must be enclosed in parentheses (). Each reader number is checked when the source file is built to ensure that the reader exists. If it does not, the build process will terminate with an error. Examples are:

```
Unlock(1)
Unlockm(25)
Relock(DevId)
Lkouton(3)
Lkoutoff(7)
```

Elevator readers are a special case, as they require additional floor information. The commands Elevunlock and Elevrelock were created to address this need. Both commands require a logical elevator reader number and a relay number (floor) to act upon. Two examples are shown below:

```
Elevunlock(3, 5)
Elevrelock(7, 49)
```

Similar to the reader commands, but affecting all defined readers as a group, is the global anti-passback command. If global anti-passback levels are defined within the system, global anti-passback processing may be enabled or disabled using the

command GlobalAPB. This command is followed by an ON or OFF keyword in parenthesis, which indicates the type of action to take. Examples are:

```
GlobalAPB(ON)
GlobalAPB(OFF)
```

Input points may be shunted or shunts removed via these three commands: Shunt, Shuntm, Unshunt. Shunt and Unshunt apply and remove a global shunt to the specified input point. Shuntm, like Unlockm applies a 10 second global shunt. Examples would be:

```
SHUNT(DevId)
SHUNTM(129)
UNSHUNT(1024)
```

As with readers and inputs, outputs can be controlled with the three commands Activate, Activatem and Deactivate. Examples are:

```
ACTIVATE(12)
ACTIVATEM(1999)
DEACTIVATE(DevId)
```

Device State Command Expressions

The macro language provides expressions that permit a macro author to test for individual device states and make decisions based on those states. In order to perform this logic, a means by which to test a device for a specific state is needed. In the following topic, we examine the methods available for testing device states.

An input point device may at any time be in one the following states: alarm, secure, trouble or offline. The macro language provides an expression for retrieving the current state of an input point. The syntax is presented below:

Input(**nnn**, **state**)

where **nnn** represents a valid logical input point number, and state is one of the following character strings: **ALARM, SECURE, TROUBLE,** or **OFFLINE**. For example, assume that 145 is the id number of an input, the expression:

```
Input(145, ALARM)
```

will examine the current state of input 145 and return the value **TRUE** if the input is currently in alarm, and **FALSE** otherwise. Likewise the expression:

```
Input(145, SECURE)
```

will return **TRUE** if input 145 is currently in a secure state, and **FALSE** otherwise.

Similar expressions are available to test the states of readers and output points. Examples of reader expressions are:

```
Reader(10, LOCKED)
Reader(10, FORCED)
```

Output examples are:

```
Output(146, ON)
Output(146, OFF)
```

The **TRUE** or **FALSE** values these expressions return form the basis of the decision making capability built into the macro command language. The flow control expressions discussed next use these expressions of true or false to direct the execution of a macro program.

Text and Data Commands

Serial and Parallel Output

It may be necessary to log actions to a printer, or send special command sequences to serial devices. The commands Print, Printbyte, SetBaud, Send, and SendByte commands accomplish this. A Print command is displayed below:

```
PRINT(1, “These words are printed on LPT1")
```

The first number 1 specifies the printer port to print the message on. This may be 1, 2, or 3 corresponding to printers LPT1, LPT2, or LPT3. The second message portion of the command “These words are printed on LPT1" comprises the actual data to be printed on the printer. This message must be enclosed in double quotes to delimit the message.

Although we have printed our message, we will probably want to continue printing on the next line at some point. We can use the PrintByte command to instruct the printer to move to the next line as follows:

```
PrintByte(1, 13)
PrintByte(1, 10)
```

In this example, we send the printer a carriage return code 13, and the linefeed code 10 to perform this.

The Send and SendByte commands are almost identical, as shown below:

```
Send(2, “These words are transmitted out COM2")
SendByte(2, 13)
SendByte(2, 10)
```

The number 2 in this example specifies serial port COM2, while the text “These words are transmitted out COM2" comprises the data to transmit.

The communication parameters default to 9600 baud, 8 data bits, no parity and 1 stop bit. The actual baud rate can be varied through use of the SetBaud command. This command accepts a port number and a baud rate parameter which must be one of 9600, 4800, 2400, 1200 or 300. The example above can be changed to transmit at 300 baud as follows:

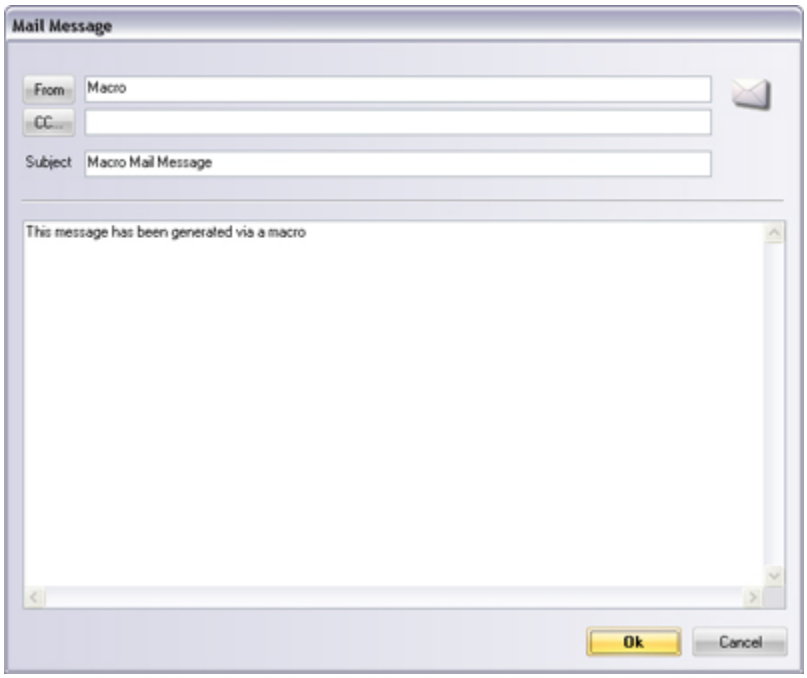
```
SetBaud(2, 300)
Send(2, “These words are transmitted out COM2")
SendByte(2, 13)
SendByte(2, 10)
```

Hurricane Mail

Macros can interact with operators by sending messages via the Hurricane Mail messaging system. The SendMail command takes three parameters: the recipients’ audit name as defined in their operator account, a message subject and finally the message text. For example the following macro command:

```
SendMail("James", "Macro Mail Message", "This message has been generated via a macro")
```

would result in the mail message below being queued into the operator’s inbox.



Counters

Counter Commands

Any macro may reference any of the 100 globally available counters. A counter is a persistent storage place for numeric values available while the Hurricane server is executing. These storage places and the values they hold persist after a macro terminates. Counters are typically used for counting things such as the number of people who have entered a room, or the number of cars in a parking lot. They could also be used for signalling purposes where a value of 0 represents false and a non-zero value equals true.

Three commands are available for manipulating counters. These are the Set, Add, and Sub commands. Each command takes as an argument, a counter id number expressed as $n, where n is a number between 1 and 100. The Set command sets a counter to a specified value. For example the statement, Set($1, 0) sets counter number 1 to the value zero. The Add command performs addition on counters. The statement Add($5, 1) adds the value 1 to the current value of counter 5. Likewise, the Sub command subtracts values from counters. The command Sub($3, 5) subtracts the value 5 from counter 3.

Counter Test Expressions

Five counter expressions are available for evaluating the current values of both counters as well as the runtime variables DevId and CardId.

The first Eq tests for equality between a counter or runtime variable and another integer, counter or runtime variable. Consider the example below:

```
EQ($1, 5)
EQ($1, $2)
```

The first expression tests for equality between counter 1 and the value 5. The second expression tests for equality between counter 1 and counter 2. Like device state expression, counter expressions evaluate to true or false and can be used in flow control statements.

```
EQ(CardId, 100)
EQ(DevId, $1)
```

In the two examples above, runtime variables are demonstrated as parameters. In the first, the runtime variable CardId is tested to see if it is equal to the value 100. This would be true if access card 100 was associated with execution of the macro. The second example tests if the runtime variable DevId is equal to the value stored in counter 1.

The remaining counter expressions are listed below:

Gt	Greater than
Ge	Greater than or equal
Lt	Less than
Le	Less than or equal

Flow Control

Flow control statements are what set the macro command language apart from the simple device state to command linking available under Hurricane. Flow control commands allow the macro author to make decisions and take different actions based on the result of expressions discussed previously. This magic is accomplished with three basic commands, Repeat, While and If Else. Each statement permits control of the order or flow of macro command execution by Hurricane. We will examine each of the different statements next.

The simplest of these commands is the Repeat statement. It allows a set of one or more commands or statements to be executed a fixed number of times. Consider the following:

```
Repeat(5)
        Activatem(25)
End
```

This small command script performs 5 momentary output activations to output point 25. The basic structure is the statement Repeat(*n*), where *n* is a digit from 1 to 1000.

This statement is followed by the commands or statements to be repeated. In this example, it is the single command Activatem. Notice the indentation on this line. This is a common technique for indicating that the preceding command or statement is under the control of this one. Our example ends with the End statement. This statement serves as the end marker for our block of Repeat commands. Remember that more than one statement can be included, as in below:

```
Repeat(10)
        Unlockm(1)
        Unlockm(2)
        Activatem(10)
        Shuntm(11)
End
```

The next flow control command is the While statement. It is the first statement to exploit the expressions we discussed earlier. A simple example follows:

```
While Input(11, ALARM)
        Activate(10)
        Deactivate(10)
End
```

This small script activates and deactivates output point 10 as long as input point 11 remains in alarm. It does this by continually checking the state of input point 11 in the While statement itself. Again the End statement marks the block of statements controlled by the While.

The If statement serves as the decision making command. If permits us to check the state of any device and if it is in that state (**TRUE**), then perform the block of statements that follow. Consider the example below:

```
If Reader(1, TAMPER)
        Activate(10)
End
```

The Activate command is only executed if reader 1 is currently in a tamper state. If it is not, the flow or order of statements switches to statement(s) following the End. Teamed up with Else statement, If allows the classic if then else logic to be implemented as shown below:

```
If Reader(1, TAMPER)
        Activate(10)
Else
        Deactivate(10)
End
```

If reader 1 is in a **TAMPER** state, then Activate output 10, Else Deactivate output point 10.

Now we are not restricted to the simplicity of the preceding examples, we can combine the discussed flow control statements by making them blocks of preceding statements as follows:

```
While Input(11, ALARM)
        If Reader(1, TAMPER)
                Activate(10)
        Else
                Deactivate(10)
        End
End
```

Notice how the If group of statements serves as the statement block to the While statement. This condition is known as nesting. Several layers of nesting are possible when constructing your macro command files.

Language Reference

Device Commands

Card Readers

Unlock(id)
Maintained unlock of reader specified by logical reader id.

Unlockm(id)
Temporary unlock of reader specified by logical reader id.

Relock(id)
Re-lock a reader specified by logical reader number id.

ElevUnlock(id, relay)
Unlock the floor controlled by relay specified by logical reader id.

ElevRelock(id, relay)
Re-lock the floor controlled by relay specified by logical reader id.

GlobalAPB(ON/OFF)
Turn global anti-passback processing on or off.

Outputs

Activate(id)
Turn on the output point specified by logical point number id.

Activatem(id)
Turn on the output point specified by logical point number id for 10 seconds.

Deactivate(id)
Turn off the output point specified by logical point number id.

Inputs

Shunt(id)
Shunt the input point specified by logical point number id.

Shuntm(id)
Shunt the input point specified by logical point number id for 10 seconds.

Unshunt(id)
Remove a shunt from the input point specified by logical point number id.

Text and Data Commands

Parallel Device Output

Print(port, “message”)
Print a message on the printer attached to port. The values for port may range from 1 for LPT1 to 3 for LPT3. The message may consist of any printable characters enclosed between double quotes.

PrintByte(port, code)
Send an ASCII code to the printer attached to port. The values for port may range from 1 for LPT1 to 3 for LPT3. The code value may be any value from 0 to 255.

Serial Device Output

SetBaud(port, baudrate)
Change the default baud rate for port from 9600 to the value baudrate. Permissable values for baudrate are 9600, 4800, 2400, 1200, and 300. SETBAUD must be called before the first transmission out of the port to be effective.

SetStopBits(port, stopbits)
Change the default number of stop bits from 1 to the value stopbits. Permissable values for stopbits are 0 for 1 stop bit, 1 for 1.5 stop bits, and 2 for 2 stop bits. SETSTOPBITS must be called before the first transmission out of the port to be effective.

SetParity(port, parity)
Change the default parity from none to the value parity. Permissable values for parity are 0 for none, 1 for odd, 2 for even, 3 for mark and 4 for space parity. SETPARITY must be called before the first transmission out of the port to be effective.

Send(port, “message”)
Transmit a message out serial port. The values for port may range from 1 for COM1 to the highest port number available on your computer. The message may consist of any characters enclosed between double quotes.

SendByte(port, code)
Send an ASCII code out serial port. The values for port may range from 1 for COM1 to the highest port number available on your computer. The code value may be any value from 0 to 255.

Mail Messaging

SendMail("to", "subject", "message")
Send a Hurricane mail message to the operator with audit name "to", with the subject line "subject" and the message specified by "message".

Counter Commands

Set(Counter, Value)
Set the counter designated by Counter to the value specified by Value.

Add(Counter, Value)
Add the value specified by Value to the current contents of the counter specified by Counter.

Sub(Counter, Value)
Subtract the value specified by Value from the current contents of the counter specified by Counter.

Expressions

Device Expressions

Reader(*id*, *state*)

If the status of the reader with logical number *id* is equal to *state*, evaluate to **TRUE**, else evaluate to **FALSE**. Valid states are:

LOCKED	**TRUE** if reader is currently locked
UNLOCKED	**TRUE** if reader is currently unlocked
SYSCODE	**TRUE** if reader is currently in system code mode
DUAL	**TRUE** if reader is currently in dual custody mode
GAPB	**TRUE** if reader is currently in global anti-passback mode
FORCED	**TRUE** if a forced entry alarm is present
TAMPER	**TRUE** if a tamper alarm is present at the reader
DHO	**TRUE** is a door held open alarm is present at the reader
OFFLINE	**TRUE** if the reader communications have failed

Input(id, state)

If the status of the input point with logical number id is equal to state, evaluate to **TRUE**, else evaluate to **FALSE**. Valid states are:

ALARM	**TRUE** if the input point is currently in an alarm state
SECURE	**TRUE** if the input point status is currently secure
TROUBLE	**TRUE** if the input point currently has trouble status
SHUNTED	**TRUE** if the input point is currently shunted
OFFLINE	**TRUE** if the input point device's communications have failed

Output(id, state)

If the status of the output point with logical number id is equal to state, evaluate to true, else evaluate to false. Valid states are;

ON	**TRUE** if the output point is currently activated.
OFF	**TRUE** if the output point is currently deactivated.
OFFLINE	**TRUE** if the output point device's communications have failed.

Counter Expressions

Eq(*Counter1 or Runtime Variable*, *Value* or *Counter2)*
If the counter specified by *Counter1 or Runtime Variable* is equal to the value specified by *Value* or the contents of the counter specified by *Counter2*, evaluate to TRUE, else evaluate to FALSE.

Lt(*Counter1 or Runtime Variable*, *Value* or *Counter2*)
If the counter specified by *Counter1 or Runtime Variable* is less than the value specified by *Value* or the contents of the counter specified by *Counter2*, evaluate to TRUE, else evaluate to FALSE.

Le(*Counter1 or Runtime Variable*, *Value* or *Counter2*)
If the counter specified by *Counter1 or Runtime Variable* is less than or equal to the value specified by *Value* or the contents of the counter specified by *Counter2*, evaluate to TRUE, else evaluate to FALSE.

Gt(*Counter1 or Runtime Variable*, *Value* or *Counter2*)
If the counter specified by *Counter1 or Runtime Variable* is greater than the value specified by *Value* or the contents of the counter specified by *Counter2*, evaluate to TRUE, else evaluate to FALSE.

Ge(*Counter1 or Runtime Variable*, *Value* or *Counter2*)
If the counter specified by *Counter1 or Runtime Variable* is greater than or equal to the value specified by *Value* or the contents of the counter specified by *Counter2*, evaluate to TRUE, else evaluate to FALSE.

Flow Control

Repeat *nnn*
    statements
End

Repeat the block of statements between the Repeat and End statements *nnn* times.

While *expression*
    statements
End

Repeat the block of statements between the While and End commands while the expression is true.

If *expression*
    statements
End

Execute the block of statements between the If and End statements only if the *expression* is true. Otherwise, jump ahead to the statements following the End statement.

If *expression*
    statements1
Else
    statements2
End

If the *expression* is true, execute statements1 and jump to the statements following the End statement when the Else statement is reached. If the *expression* is false, execute statements2.