Alpha® Marquee Control

ActiveX Developer's Reference

Version 1.1

Table of Contents

INTRODUCTION

- The Marquee Control is a development tool designed to ease communication with an Alpha Marquee Display.

- The Marquee Control is an ActiveX Control. As such, it should function in any container that supports the ActiveX standard.

- The Marquee Control can display predefined messages, stored in 'Message Files' in XML format, or messages created at run-time.

- The Message Manager utility is provided with the Control to allow message creation, modification, and deletion. A given message file can hold as many messages as desired.

- The Marquee Control offers multiple modes of operation for displaying different types of data.

SYSTEM REQUIREMENTS

Microsoft Internet Explorer 5.0 (or greater)

> Internet Explorer must be installed on the machine, but need not be the default browser. The Control simply utilizes libraries that are installed with Internet Explorer.
>
> Internet Explorer 4.01 can meet this requirement if the 'xmlredist.exe' is used to upgrade the needed libraries. This file is run automatically upon installation if needed. If Internet Explorer 4.01 is installed after the Control, 'xmlredist.exe' must be run before the Control will function properly.

A design environment that supports ActiveX controls. The Control should function in any ActiveX container, such as Visual C++; compatible manufacturing software/Human Machine Interfaces (HMI), as well as business applications such as Microsoft Excel.

NOTE: The following five containers have all been thoroughly tested and message samples for them are available, along with telephone support:

- Microsoft Visual Basic 6.0 SP3
- GE Cimplicity 4.01 SP3
- Intellution iFix 2.1
- Rockwell RSView32 6.20.49
- Wonderware Intouch 7.1

A serial or LAN connection to an Adaptive Alpha Marquee.

GETTING STARTED

Open the container to be used in its design environment.

Load / Install the Control within the container if needed.
For example, in Visual Basic, the Control must be selected under "Components", or in Wonderware's Intouch, an ActiveX Control must be installed with the Installation Wizard.

Place an instance of the Control on the application.

Bring up the Control's property pages. Set properties to desired values.
Minimally: set **MarqueeAddress** on the 'General' tab, and set LAN or Serial properties on the Communication tab.

Place code in a script/event/function to call the **Initialize** method upon startup.

As desired, add code to call methods and set properties of the Control.

Run application.

GENERAL USE

Initialize the Control

The **Initialize** method should be called before any other method on the Control. Startup scripts/events of the containing application are a good place to do this.

Pre-defined Messages

Calling **ChangeMode** makes use of other properties on the Control. Make sure to set these properties before calling **ChangeMode**.

For example, to display a template, the **TemplateName** and **MessageFile** properties should combine to reference an existing template in an existing file. Thus, in Visual Basic:

```
Marquee1.MessageFile = "c:\AMS Marquee\Demos\demoMessages.xml"
Marquee1.TemplateName = "Status"
Marquee1.ChangeMode("String")
```

If one file contains all the templates to be used by the application, **MessageFile** can be set at design time on the 'General' property page. Then, the property does not need to be set through code.

If the message contains variables, those variables can be updated with a call to the **UpdateVariable** method.

Custom Messages

Messages can be created through code with a single call to the **QuickDisplay** method**.** Attributes of the message are sent as parameters to the method.

Other functionality

The **ResetBoard** method can be used to send a reset signal to the marquee. This causes the marquee to run through its startup routine. The reset is a "soft" reset that is non-destructive to the marquee's internal memory.

The **Clear** method can be used to remove all messages from the marquee.

MODES

The Control can function in several different modes.

Template Mode

Template mode is designed for displaying production information or statistics. Templates are designed to fill the marquee display and do not move on and off the marquee. Thus only color (per character) and font (per template) attributes are available.

Parts of the template can be configured statically to display labels, such as "Machine#" or "Parts/Hr". Other parts of the template can be configured to show variables, like the current value of a machine number or parts per hour. Variables can be updated at any time using the **UpdateVariable** method.

Example:

Parts/Hr: XXXX Count: XXXX
Machine #: XX

The Message Manager utility is used to design Template messages.

String Mode

String mode is intended for use with messages that have movement. Color, font, mode, position, speed and special attributes are available for strings. Color can be selected on a character basis; remaining attributes affect the complete string.

Example: "Shift 2 has met its production goal!" can be configured to scroll left to right, at low speed, across the middle of the marquee, with various colors, in five high standard letters.

String mode supports variables, which are updated by calls to the **UpdateVariable** method. Therefore, the above example could be easily be used for shift 3 or 4 by making the shift number a variable.

The Message Manager utility is used to design String messages.

Alarm Mode

Alarms are Strings that must be acknowledged before other messages can be displayed. However, unlike Strings, Alarms do not support the ability to define variables.

The Control maintains an internal queue of alarms. All alarms in the queue are displayed one after another in a round-robin manner

Alarm mode can be entered explicitly or implicitly.

- Calling the **ChangeMode** function will explicitly put the marquee in alarm mode. If no alarms are in the Control's queue, nothing will be displayed on the marquee. Alarms can be added and acknowledged as needed with the **AlarmAdd**, **AlarmAcknowledge**, and **AlarmAcknowledgeAll** functions. When all alarms have been acknowledged, the marquee displays nothing. The marquee will remain in alarm mode until a call to **ChangeMode** switches the current mode.

- Adding an alarm with the **AlarmAdd** function will implicitly put the marquee in alarm mode. If the Control is already in alarm mode, the alarm is added to the Control's queue. If the Control is not in alarm mode, it will save its settings, enter alarm mode, and add the first

alarm to its queue. Additional alarms can be added; any alarm can be acknowledged. Once the last alarm has been acknowledged, the Control will revert to its saved settings (displaying what it was before entering alarm mode.)

No State

The Control does not have to keep an internal state. The Control can simply be used as an interface to the marquee if desired. This can be accomplished by using only the **QuickDisplay** and Strings that contain no variables.

Alarm mode requires the Control to keep an internal queue of alarms to rotate to the marquee. Templates and Strings with variables require the Control to keep an active list of available variables and where they are located. Thus, these modes cannot be used in a stateless manner.

CONNECTIONS

Communication

Serial
Each Control is configurable at design time to use a specific COM port. The COM port is opened and closed as needed by the Control. Thus, multiple Controls can be configured to use the same COM port.

TCP/IP
Each Control is configurable at design time to use a specific remote IP address and remote port. Due to the overhead of creating a socket connection with the ethernet adapter, the socket connection is made when the Control is initialized, and terminated when the Control terminates.

Note the difference from Serial communication. Since a Control maintains the socket connection to the ethernet adapter, only one Control can use a given IP address.

Single marquee per connection

If each marquee has its own IP address or COM port, the application simply needs one Control per marquee.

Multiple marquees per connection

It is possible to "chain" marquees together. In this situation, from the perspective of the local computer, multiple marquees exist at one connection (be it a COM port or IP address.)

Serial
When multiple marquees exist behind one COM port, the number of Controls needed depends on how the marquees are to be used.

If all the marquees are to display the same data, they can be thought of as one marquee. Thus, only one Control is needed. Simply set the **MarqueeAddress** property to '00' to broadcast to all marquees.

If the marquees are to display different data, one Control per marquee will be needed. Configure each Control to use the given COM port, and set the **MarqueeAddress** property to the address of a single marquee.

TCP/IP
When multiple marquees exist at one IP address, less flexibility exists. Only one Control can access the IP address since the Control maintains a socket connection to the adapter, as discussed above.

If all the marquees are to display the same data, they can be thought of as one marquee. Thus, only one Control is needed. Simply set the Control's **MarqueeAddress** property to '00' to broadcast to all marquees. This is the same as a Serial connection.

The difference becomes apparent when the marquees are to display different data. An instance of the Control has only one internal state. Since the marquees are to display different data, multiple Controls are required. However, only one Control can use the single IP address. Thus, if the Control monitors state (Templates, Strings with variables, or Alarms), all marquees at the IP address must display the same data.

If the Control can be used in a "stateless" mode, the marquees can display different data. Prior to any communication with the marquees, simply reset the **MarqueeAddress** property to the desired marquee. Stateless operation means using only **QuickDisplay** and Strings with no variables.

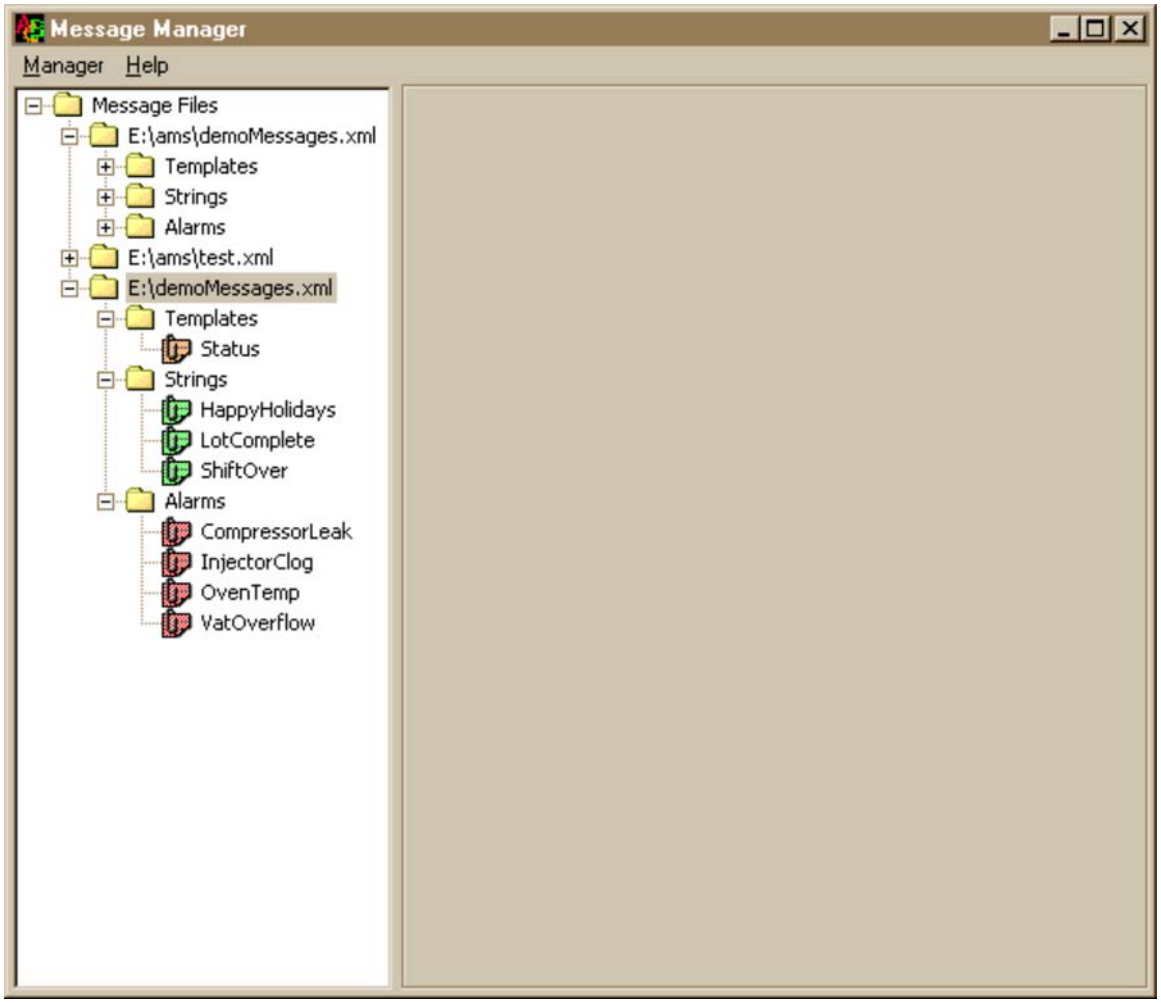
MESSAGE MANAGER UTILITY

Templates, Strings, and Alarms are stored in message files to be used by the Control. The messages are stored in XML format, hence the ".xml" extension. The Message Manager utility is installed in the same directory as the Marquee Control. A sample message file, "demoMessages.xml" is installed in the 'Demos' directory.

Message files can contain any number of Alarms, Templates, and Strings.

User Interface

The Message Manager utility provides a user interface for creating the Message files. The main window is divided into two panes.

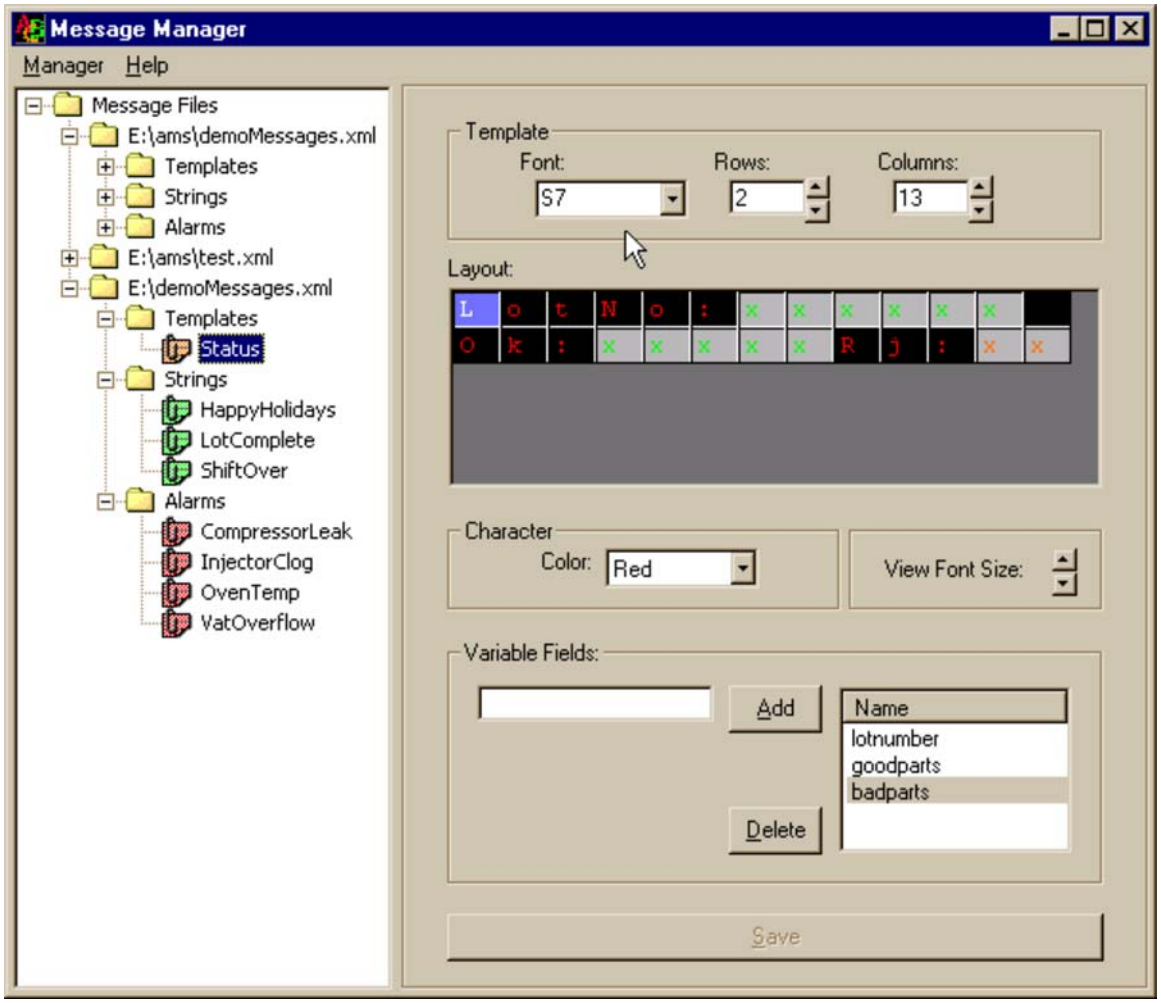


The left pane operates much like the standard Windows Explorer. Message files can be added and removed as desired. When a message file is loaded, the contained messages are grouped by type. When a single message is selected, an editor should appear in the right pane and load the selected message.

The right pane displays an editor specific to the type of message selected. Templates have their own editor while Strings and Alarms share a second editor.

Templates

Selecting a Template from the tree causes the Template editor to be displayed on the right.



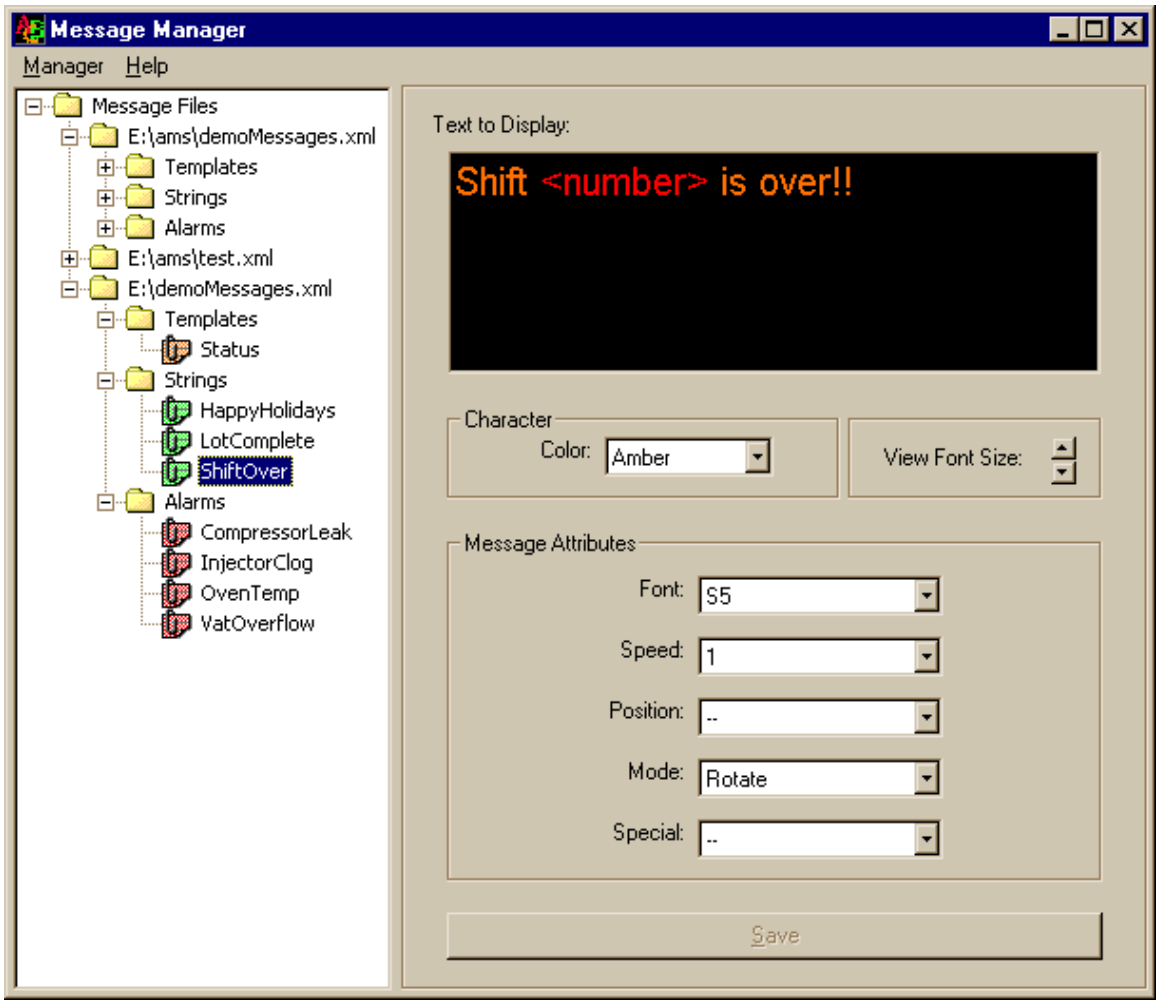
- The controls at the top determine template attributes. Rows and Columns determine the size of the grid just below. Font determines what font will be used on the marquee. This selection has no visual implications on the grid.
- The grid represents the marquee.
- Color will change the color of the cursor at its current location. Selecting several cells or a variable will change the color of all selected cells.
- View Font Size only adjusts the size of the font used in the grid. This is for user convenience only and has no effect on the message displayed on the marquee.

- Variables can be defined by selecting cell(s) on the grid, typing a name in the Variables section at the bottom, and clicking Add. The list box shows all defined variables for the Template. The variable's name is used in later calls to **UpdateVariable**.

- The Save button at the bottom saves the Template to the file.

Strings

Selecting a String message causes the String editor to be displayed on the right.



- There is a large text box at the top of the String editor. Type the message to be displayed here. Variables are created by surrounding the variable name with angle brackets. The variable's name is later used in calls to **UpdateVariable**.

- Color will change the color of any selected text in the text box.

- View Font Size only changes the size of the text in the editor. It has no effect on how the String is displayed on the marquee.

- The Save button at the bottom saves the String to the file.

- The Attributes section defines attributes for the whole message. Check the marquee's documentation for an explanation of the attributes.

- Not all values are available for every model of Alpha sign. The utility displays all values that are available. Check the marquee's documentation for a list of which values are supported. Most marquees will use a default value if an unsupported value is specified.

Alarms

Selecting an Alarm message brings up the Alarm editor on the right.

- The same editor used by Strings is used for Alarms. All attributes work in the same manner.

- The only difference from Strings is that Alarms do NOT support variables. Text surrounded by angle brackets defines a variable in Strings. In Alarms the angle brackets are considered part of the message.

METHODS

AlarmAdd

Syntax:

Marquee.AlarmAdd

Remarks:

This method is used to add an alarm message to the Control's internal alarm queue. The method attempts to add the alarm specified by the **AlarmName** and **MessageFile** properties. A specified alarm can only be added to the queue once.

Alarms in the queue are cycled to the marquee in a round-robin manner. New alarms are added to the front of the queue.

Method returns a Boolean value indicating success or failure of the method.

See Also:

MessageFile, AlarmName properties.

AlarmAcknowledge

Syntax:

Marquee.AlarmAcknowledge

Remarks:

This method is used to remove an alarm message from the Control's internal alarm queue. The method attempts to remove the alarm specified by the **AlarmName** and **MessageFile** properties.

If the alarm removal empties the internal alarm queue, and Alarm mode was entered implicitly, the Control will attempt to revert to the saved mode. See section on Alarm mode in User's guide.

Method returns a Boolean value indicating success or failure of the method.

See Also:

MessageFile, **AlarmName** properties.

AlarmAcknowledgeAll

Syntax:

Marquee.AlarmAcknowledgeAll

Remarks:

This method is used to remove all alarm messages from the Control's internal alarm queue. It is equivalent to calling **AlarmAcknowledge** for all alarms in the queue.

Method returns a Boolean value indicating success or failure of the method.

ChangeMode

Syntax:

Marquee.ChangeMode(*NewMode*)

Parameters:

NewMode – String

Indicates new mode. Modes can be referenced either by name or number.
Possible values:

Template Mode:	"Template" or "0"
String Mode:	"String" or "1"
Alarm Mode:	"Alarm" or "2"

Remarks:

This method is used to change the internal state of the Control. Calling **ChangeMode** attempts to load the message specified by the File and Name properties of the specified mode.
For example, calling **ChangeMode**("1") will attempt to change to string mode and display the string specified by **StringName** and **MessageFile**.

Note that **ChangeMode** is also used to change the message displayed without changing modes. If the Control is in String mode, a subsequent call to **ChangeMode**("string") will simply change the String displayed.

Method returns a Boolean value indicating success or failure of the method.

Clear

Syntax:

Marquee.Clear

Remarks:

This method clears the marquee display, and erases the values of all variables stored in the marquee's memory.

Does not rewrite the internal memory configuration table of the marquee.

Method returns a Boolean value indicating success or failure of the method.

Initialize

Syntax:

Marquee.Initialize(*VariableSize*)

Parameters:

VariableSize – Optional Integer

Specifies the size of the string files created in the marquee's memory. Default size (if parameter is omitted) is 32 bytes.

Remarks:

Method must be called prior to any other calling any other method on the Control.

Method creates the permanent socket connection with the marquee, if LAN connection is specified. The memory configuration table is then sent to the marquee. Configuration consists of 64 string files and one text file. Size of the string files is determined by the *VariableSize* parameter. Any remaining memory is assigned to the single text file.

Method returns a Boolean value indicating success or failure of the method.

QuickDisplay

Syntax:

Marquee.QuickDisplay(*Text*, *Color*, *Font*, *Mode*, *Special*, *Position*)

Parameters:

Text – String

The text to be displayed.

Color – Optional String

The color to use when displaying *Text*. Must be one of:

"AutoColor" – default if *Color* is unspecified
"Red"
"Green"
"Amber"
"DimRed"
"DimGreen"
"Brown"
"Orange"
"Yellow"
"Rainbow1"
"Rainbow2"
"Mix"

Font – Optional String

The font to use when displaying *Text*. Must be one of:

"S5"
"S7" – default if *Font* is unspecified
"F7"
"S10"
"FFULL"
"SFULL"

Note: Values are of the form "S(tandard) | F(ancy)" followed by the height.

Mode – Optional String

The mode in which to display *Text*. Must be one of:

"Rotate"
"Hold"
"Flash"
"RollUp"
"RollDown"
"RollLeft"
"RollRight"
"WipeUp"
"WipeDown"
"WipeLeft"
"WipeRight"
"Scroll"
"AutoMode" – default if *Mode* and *Special* are unspecified
"RollIn"
"RollOut"
"WipeIn"

"WipeOut"
"CompressedRotate"

Note: see marquee documentation for descriptions.

Special – Optional String

The special mode in which to display *Text*. *Special* overrides *Mode* if both are specified. Must be one of:

"Twinkle"
"Sparkle"
"Snow"
"Interlock"
"Switch"
"Slide"
"Spray"
"Starburst"

Note: see marquee documentation for descriptions.

Position – Optional String

The position at which to display *Text.* Must be one of:

"Top"
"Middle"
"Bottom"
"Fill" – default if *Position* is unspecified

Remarks:

Method displays the message defined by its parameters, IF POSSIBLE. Not all marquees are capable of all listed colors, modes, and fonts. Certain positions do not allow certain fonts, etc. The marquee will usually use a default value if the specified option or combination is invalid. The defaults shown above are the Control's defaults. These defaults will be sent when a parameter is not specified.

Using **QuickDisplay** does not change the Control's state. A subsequent call to **ChangeMode** will return the marquee to the mode used before **QuickDisplay**, provided none of the properties have been changed.

Method returns a Boolean value indicating success or failure of the method.

ResetBoard

Syntax:

Marquee.ResetBoard

Remarks:

Method causes the marquee to run through its startup diagnostic sequence (soft reset.) The soft reset does not clear the marquee's memory.

This method has no affect on the Control or its internal data.

Method returns a Boolean value indicating success or failure of the method.

UpdateVariable

Syntax:

Marquee.UpdateVariable(*Name, Value*)

Parameters:

Name – String

The name of the variable to be updated.

Value – String

The value to be displayed.

Remarks:

Method causes the Control to look through the list of available variables for the current message. If *Name* is found, *Value* is sent to the marquee. If *Value* is longer than the variable's defined length, *Value* is truncated. If *Value* is shorter, it is left aligned in the defined space.

Method returns a boolean value indicating success or failure of the method.

PROPERTIES

AlarmName

Type: String

Availability: Run-time only

Remarks:

Name of alarm to look for in the specified message file. Used in conjunction with **MessageFile**. This property should be set before a call to **AlarmAdd**, **AlarmAcknowledge**, or a call to **ChangeMode** that changes to Alarm mode

MarqueeAddress

Type: String (length of 2): Hexadecimal value.

Availability: Design-time & Run-time

Remarks:

Specifies the internal address of the marquee. Can be used to route a message to a specific marquee.

MarqueeConnectMode

Type: Byte [0 – 1]
0 – Serial Connection
1 – LAN Connection

Availability: Design-time only

Remarks:

Specifies the means for connecting to the marquee. Serial Connections are established and relinquished as needed. LAN Connections are established and relinquished upon Control initialization and termination.

MarqueeLogFile

Type: String

Availability: Design-time & Run-time

Remarks:

Specifies the file and path to log errors to. Any method which returns FALSE will write an error message to this file, if it is specified.

MarqueeType

Type: Byte: [0]
0 – Alpha

Availability: Design-time only

Remarks:

Reserved for future use.

MessageFile

Type: String

Availability: Design-time & Run-time

Remarks:

Message file to access when looking for a given message. Used in conjunction with **TemplateName, StringName, AlarmName**. Can be changed at run-time to allow multiple files to be used.

Can be set initially on the 'General' property page. If all messages are contained in one file and the property is set at design time, this property need not be used in code.

NetworkAddress

Type: String

Availability: Design-time only

Remarks:

Network address where LAN adapter resides. Can be an IP address, or network name, provided that the local computer can resolve it.

NetworkPort

Type: Integer: [0 – 32767]. Default is 3001.

Availability: Design-time only

Remarks:

Port at which to open socket connection with LAN adapter.

SerialComPort

Type: Byte: [1-16]

Availability: Design-time only

Remarks:

COM port to use for serial communication.

SerialBaudRate

Type: Integer: [100, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 56000, 57600, 115200, 128000, 256000]

Availability: Design-time only

Remarks:

Baud rate to at which to open COM port.

SerialParity

Type: String ["None", "Even", "Odd", "Mark", "Space"]

Availability: Design-time only

Remarks:

Parity used when opening COM port.

SerialDataBits

Type: Byte: [4, 5, 6, 7, 8]

Availability: Design-time only

Remarks:

Data bits used for COM port.

SerialStopBits

Type: Byte: [1, 2]

Availability: Design-time only

Remarks:

Stop bits used for COM port.

StringName

Type: String

Availability: Run-time only

Remarks:

Name of string to look for in the specified message file. Used in conjunction with **MessageFile**. This property should be set before a call to **ChangeMode** that changes to String mode.

TemplateName

Type: String

Availability: Run-time only

Remarks:

Name of template to look for in the specified message file. Used in conjunction with **MessageFile**. This property should be set before a call to **ChangeMode** that changes to Template mode.

Appendix: Version 1.1 changes

New Method: UpdateVariableAdv

- **Description**

An extended version of the **UpdateVariable** method has been added to the Control entitled **UpdateVariableAdv**. This new method retains all functionality of **UpdateVariable** and adds two optional parameters that control the variable's color and flash state.

Marquee.UpdateVaraibleAdv(*Name, Value, Color, Flash)*

- **Performance**

BEHAVIOR: Flashing a template variable decreases update speed!

EXPLANATION: By design, while showing a template, the marquee uses zero delay between updates to its display. This allows all variables on the marquee to be updated as fast as possible. However, a non-zero delay is used when flashing a variable in order to allow time for the flash to be visible. This delay causes the performance decrease.

Depending on the type of data shown, this behavior may or may not be acceptable. If real-time data is the goal, avoid flashing variables. If an update approximately every 3-5 seconds is acceptable, flashing variables are a viable option.

- **Optional Parameters**

The meaning of 'Optional' in the phrase 'Optional Parameter' depends greatly on the HMI in use.

For example, Microsoft's Visual Basic© allows:

Marquee.UpdateVaraibleAdv(*Name, Value, Color)*

Marquee.UpdateVaraibleAdv(*Name, Value, , Flash)*

In Wonderware's Intouch©, the first statement is legal, but the second statement would be:

Marquee.UpdateVaraibleAdv(*Name, Value,"", Flash)*

Consult the individual HMI's documentation complete details on optional parameter syntax.

- **Syntax**

UpdateVariableAdv

Syntax:

Marquee.UpdateVariableAdv(*Name, Value, Color, Flash*)

Parameters:

Name – String

The name of the variable to be updated.

Value – String

The value to be displayed.

Color – String

Optional. Determines the variable's color. If omitted, variable's color is determined by the message. Possible values:

"AutoColor"
"Red"
"Green"
"Amber"
"DimRed"
"DimGreen"
"Brown"
"Orange"
"Yellow"
"Rainbow1"
"Rainbow2"
"Mix"

Flash – Byte [0 – 1]

Optional. Causes variable to flash. If omitted, variable does not flash.

0 – Off (no Flash)
1 – On (Flash)

Remarks:

Method causes the Control to look through the list of available variables for the current message. If *Name* is found, *Value* is sent to the marquee.

If *Value* is longer than the variable's defined length, *Value* is truncated. If *Value* is shorter, it is left aligned in the defined space.

Since flashing only occurs while the message is stationary on the marquee, *Flash* is dependent on the mode and speed of the message that contains the variable.

If *Color* is omitted, the variable is displayed in the color defined by the message.

Method returns a boolean value indicating success or failure of the method.

Change: Communication Property Availability

In an effort to make the Control as flexible as possible, the "design-time-only" attribute of several properties has been lifted. The following properties are now modifiable through code, at run-time, until the **Initialize** method is called:

MarqueeConnectMode

NetworkAddress

NetworkPort

SerialComPort

SerialBaudRate

SerialParity

SerialDataBits

SerialStopBits

Once **Initialize** has been called there should be no need to change these properties, and thus they are not modifiable.

1) FIX: DoEvents

A) The Control calls DoEvents after each transmission in TCP/IP mode. This relieves the end-user from calling DoEvents.

B) Recompilation with Visual Studio SP4 should fix bug where multiple controls using TCP/IP communication can cause communication loss if not separated by DoEvents.

For example, if two Marquee controls were used, Marquee1 and Marquee2, sending data to both without separating them with a DoEvents usually resulted in only the second send being successful:

```
Marquee1.UpdateVariable("test", 1)
Marquee2.UpdateVariable("test", 1)          ' Clobbers the call by Marquee1

Marquee1.UpdateVariable("test", 1)
DoEvents
Marquee2.UpdateVariable("test", 1)          ' OK because of DoEvents call
```

With version 1.1, the first scenario should function properly.

Service Pack 4 for Visual Studio supposedly fixes this issue with an update to the Winsock control. Microsoft Knowledgebase Article: Q245159. (The Marquee Control uses the Winsock Control for TCP/IP communication.)

NEW: Addition of ResetCommunication method.

Serial mode: Closes COM port if stranded open.

TCP/IP mode: Closes socket connection, attempts to re-establish the socket connection. A boolean return value indicates success/failure of method call.