xAPmcsRain8Net

Rain8 Net

Distributed Computing Node

Michael McSharry
March 1, 2006

Table of Contents

Table of Figures

1 Introduction

xapmcsRain8Net is a connector that bridges the Rain8Net serial protocol to xAP on an IP interface. The xAP schemas utilized is Basic Status and Control 1.3. It is installed at any location on a PC and will use subfolders to retain configuration information, this document, and HTML formatting information.

This xAP node allows local and xAP message control of Rain8Net modules. It also allows for script control of valves or scripting based upon content of received messages.

The Rain8Net is modeled as eight controllable valves for each Rain8Net module discovered. Each end nodes is addressed by the subaddress field of an xAP message and by the UID of the xAP message. The UID is formed from the Module ID and the each of eight valves within the module. This will allow upto seven Rain8Modules which are addressed sequentially from 1 to 7.

The subaddress is formed from three parts. The first is a user-provided location. The second is a unique name fromed from the module and value numbers. The third is the type name “Rain8Net” of the node.

Two keys are used in the body of the message. The “state” key is always present and in general it contains the ON, OFF, FAIL status. The “DisplayText” key is used to provide HTML-formatted status.. When commanded via xapbsc.cmd then the ON or OFF state will be used to control the Rain8Net valve.

A fully formed typical xAP message is shown in section 1.1. Note the subaddress and UID as described above.

1.1 xAP Message Format

```
xap-header
{
    v=12
    hop=1
    uid=FF000203
    class=xapBSC.Event
    source=mcs.Rain8Net.MCS5:Back.Rain8Net___01_V03.Rain8Net
}
output.state.1
{
    state=OFF
    displaytext=<table><td><img alt='Sprinkler OFF'
src='/xapmcsRain8Net/images/SprinklerOFF.gif' height='16' width='16'><img alt=' '
src='/xapmcsRain8Net/images/sensors/blank3.gif'></td><td valign='middle'
align='left'>OFF</font></td></table>
}
```

Any change of state from a Rain8Net node will be communicated as a xapbsc.event message. .

xapbsc.cmd messages are used to change the state of a node. The new state will be commanded and when the Rain8Net updates its status a xapbsc.event message will be delivered to confirm that action has been completed.

When a xapbsc.query message is received then xapmcsRain8Net will query the Rain8Net to retrieve its current state. The state should be the same as the internally managed state, but the status request is done to assure that both are in sync. A module status message will contain the status of all the components of the module. If the xapbsc.query is wildcarded to get status of multiple devices, then xapmcsRain8Net will only send module status requests once for each unique module for which a component or device status is needed.

2 Quick Start

The following outlines the steps to achieve computer control of Rain8Net via xapmcsRain8Net and Homeseer. The Homeseer-related steps are not required as xapmcsRain8Net is a self-sufficient computing node for a pure distributed environment.

- HOMESEER
- Expand mcsXap plugin into the Homeseer folder
- Register hspi_mcsXap.ocx using regsvr32 hspi_mcsXap.ocx in the Homeseer folder
- Start Homeseer and select mcsXap as an IO interface
- Configure mcsXap via its setup to use BSC transmit and receive
- XAP
- Start an xAP hub such as xapmcsHub
- Expand xapmcsRain8Net using the folder structure encoded in the zip file
- Configure xapmcsRain8Net from tray icon to select the serial port and other preferences
- Refresh browser to view the Rain8Net nodes
- Identify via checkbox in the Used column the devices of interest, Save selections
- HOMESEER INTEGRATION
- Click the Query Rain8Net button to make the devices visible to mcsXap plugin
- From mcsXap plugin browser accept messages of Rain8Net nodes of interest
- Click the Query Rain8Net button on xapmcsRain8Net to deliver the enumeratrions for Homeseer control.

The above sequence is one of several possible configurations. Editing and customizations will naturally be peformed to tailor the setup as one’s understanding improves.

3 Tray Interface

xapmcsRain8Net runs in the Windows tray with access via an icon that looks like a rayed triangle. This icon will blink for each Rain8Net transaction. Left of right click of the icon will present several options.

Exit is used to terminate xapmcsRain8Net.

The IO Window is used to see the raw traffic between the PC and the Rain8Net.

The user interface access is from the top of menu options. The GUI interface is a basic form that may be needed to enter the HTTP port before browser access is available. Setup items can be edited in this form if desired. The same setup information can be set with the Browser interface menu option.

The system config and status selections have not been activated in xapmcsRain8Net. Either can be used to cancel the presence of the icon popup menu.

4 Browser Interface

4.1 Content Selection

The browser interface provides administrative access as well as direct status and control of the features of the Rain8Net. The setup is organized into four sections. The content of each section will vary depending upon the checkbox settings in the last section's "Browser Content Display Control" as shown in **Error! Reference source not found.**. Each checkbox selection will enable/disable inclusion of content displayed by the browser.

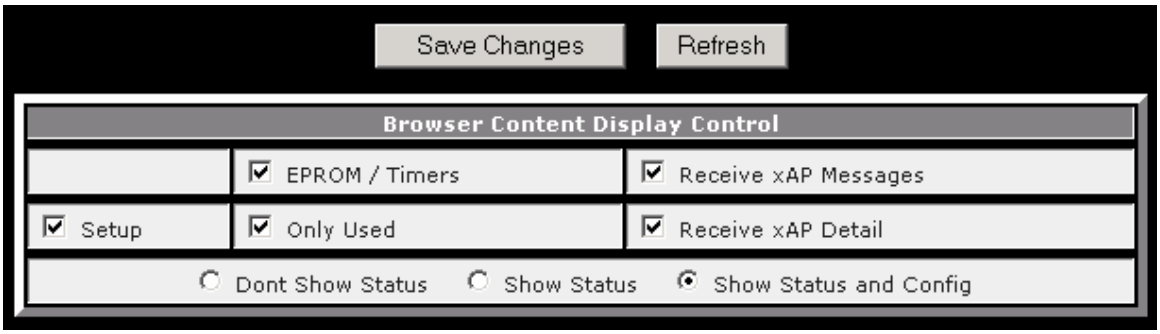


Figure 1 Browser Content Selection

4.2 Configuration

mcsxapRain8Net contains an HTTP server to provide the primary user interface. The port used by the server defaults to 8016, but can be changed via user input. See Figure 2. Most formatting is controlled by a style sheet except the body background. The default style sheet is the provided StyleNoBody.css located in the \HMTL\xapmcsRain8Net folder. This same folder contains icons as well as a file links.htm that contains links to other locations. These links are displayed under the Title bar of the browser page. The links.htm file can be deleted or edited to suit individual preferences.

Connection to the Rain8Net master is via a serial port. This port can be directly connected to the PC or connected via an IP. If directly connected then the port number is entered. If IP connected then both the IP address and port are entered.

Each xAP node has a unique ID. xapmcsRain8Net defaults to FF000200, but this dan be changed as desired.

A script can be run based upon a valve change, selected xAP message reception, or periodically. To cause \Scripts\xapmcsRain8net.txt to be executed periodically a number is entered into the Periodic Script Interval text box. The script will be called with no

parameters in this case. The event-based scripts are further described in paragraphs 4.5, 4.6, and 5.

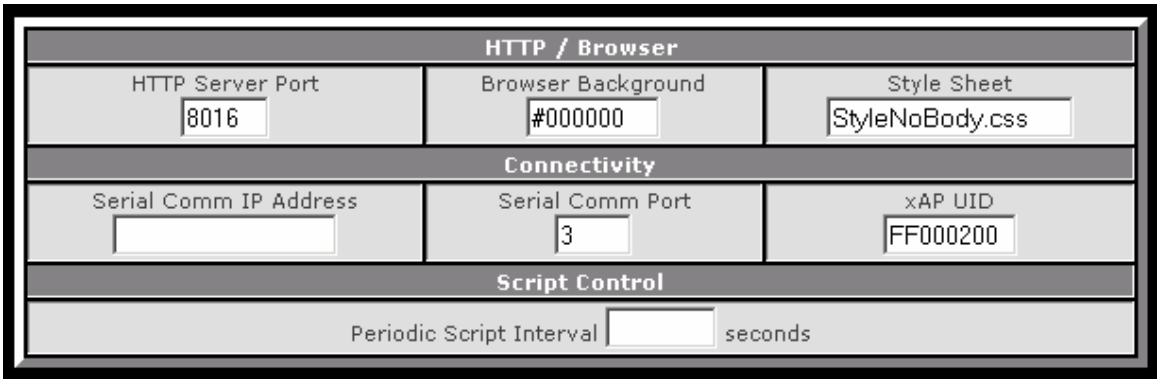


Figure 2 Node Configuration

4.3 Rain8 Net Global Control

Two controls are provided to globally address the Rain8Net. Individual value controls are described later. The "Send All-Off Command" button is used to deliver the command to the Rain8Net modules to turn all valves off.

The "Find Rain8 Modules" button is used when a module is first connected to the Rain8 Network. After the button is clicked a third cell will open next the clicked button and reflect the module number that is currently being searched. When this number is larger than the last module ID installed, then the "Abort" button can be clicked to stop the search. The result of the serarch will be displayed below the buttons and the new modules will be available for individual control.

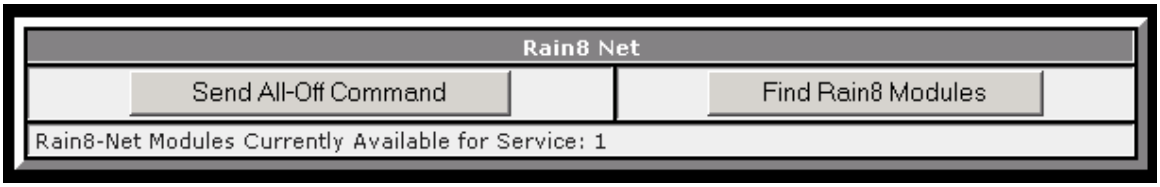


Figure 3 Rain8Net Global Controls

4.4 Rain8 Net EPROM / Timers

The Rain8Net EPROM can be configured using the selections shown in Figure 4.

The programming port is a direct connect port that may or may not be the same as the genral use port setup earlier.

The address is the module address to which the unit will respond. The Rain8Net master will typically be “1” and the slaves will be “2”, “3”, …

Timers can be enabled or disabled. If enabled then the values up to 255 can be entered for each valve.
The actual Read of the EPROM or Write of the EPROM is done withh the two buttons provided in this section.

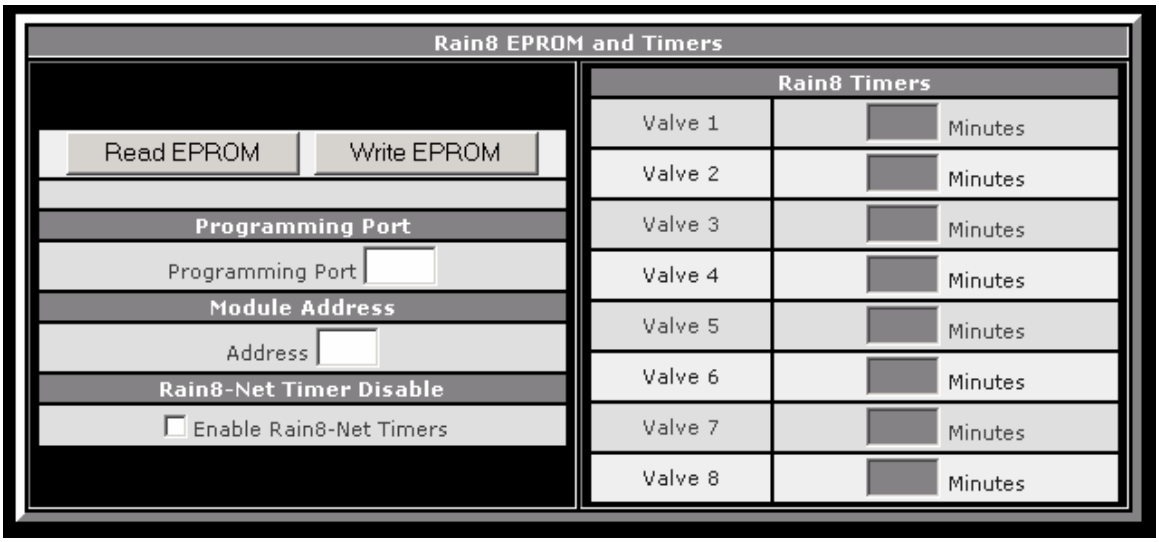


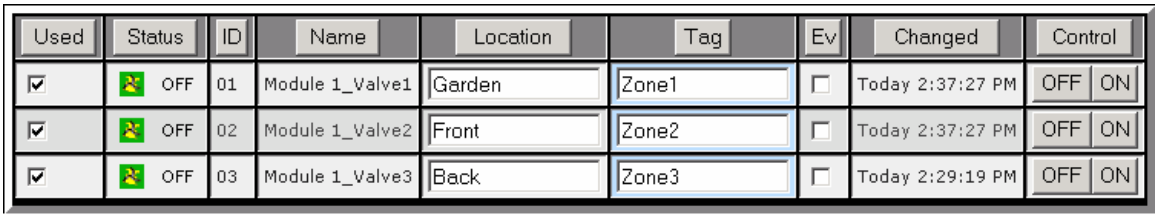
Figure 4 EPROM Programming

4.5 Rain8Net Individual Valve Control

After a module has been discovered it will be available for use. Figure 5 shows that each valve has four user configuration selections. The “Used” checkbox is used to make the valve controllable. It will also make it visible on this table when the “show only used” checkbox is used on the browser selection panel.

The “Location” entry is used to provide a name to the valve. It will included in the xAP message subaddress field.

The “Tag” and “Ev” fields are associated with scripting. The Tag value entred will be the name used when referencing the valve from a script. The Ev (event notification) checkbox is used if the script should be called when the status of the value changed. When the script is called the tag name is passed as an input parameter to the script. The actual scripting interface is described in section 5.



Used	Status	ID	Name	Location	Tag	Ev	Changed	Control
☑	OFF	01	Module 1_Valve1	Garden	Zone1	☐	Today 2:37:27 PM	OFF ON
☑	OFF	02	Module 1_Valve2	Front	Zone2	☐	Today 2:37:27 PM	OFF ON
☑	OFF	03	Module 1_Valve3	Back	Zone3	☐	Today 2:29:19 PM	OFF ON

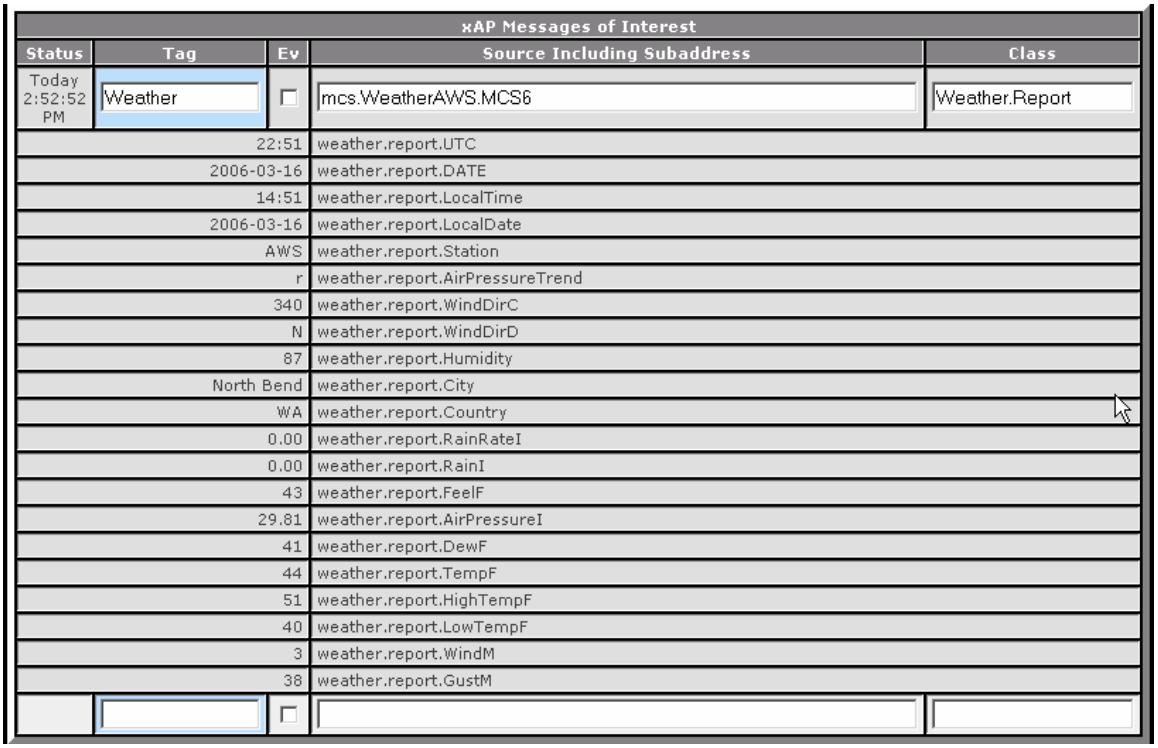
Figure 5 Rain8Net Valve Control and Status

4.6 xAP Messages of Interest

The scripting interface supports use of received xAP messages. The selection of the messags of interest is done from the panel shown in Figure 6. Four controls are provided. The frist two, "Tag" and "Ev" operate in the same manner as these same fields on Figure 5. The Tag is the name used to reference the message and Ev is used to trigger the script when the message is received. When the script is called the tag name is passed as an input parameter to the script. The actual scripting interface is described in section 5.

The address, including subaddress, of the message and the class of the message are also entred to identify the message for which data will be collected for scripting use.

When the "Receive xAP Detail" checkbox is checked then the Key/Values of the message will also be displayed. These are always available via scripting even when not displayed on the Browser.



xAP Messages of Interest				
Status	Tag	Ev	Source Including Subaddress	Class
Today 2:52:52 PM	Weather	☐	mcs.WeatherAWS.MCS6	Weather.Report
22:51			weather.report.UTC	
2006-03-16			weather.report.DATE	
14:51			weather.report.LocalTime	
2006-03-16			weather.report.LocalDate	
AWS			weather.report.Station	
r			weather.report.AirPressureTrend	
340			weather.report.WindDirC	
N			weather.report.WindDirD	
87			weather.report.Humidity	
North Bend			weather.report.City	
WA			weather.report.Country	
0.00			weather.report.RainRateI	
0.00			weather.report.RainI	
43			weather.report.FeelF	
29.81			weather.report.AirPressureI	
41			weather.report.DewF	
44			weather.report.TempF	
51			weather.report.HighTempF	
40			weather.report.LowTempF	
3			weather.report.WindM	
38			weather.report.GustM	
		☐		

Figure 6 xAP Scripting Messages Of Interest

4.7 Refresh

Two control buttons are provided to simplity integration of xapmcsRain8Net as shown in Figure 7. The “Query Rain8Net Modules” button is used to have xapmcsRain8Net deliver xapbsc.info message for all the used valves. This is useful when integrating with something like Homeseer and devices discovery is needed to create the Homeseer devices.
The “Read Script xapmcsRain8nete.txt” button is used to reread the file \Script\xapmcsRain8Net.txt. This is the script file that is executed based upon periodic or triggered events. This allows the file to be edited and when edits are complete the updated version can then be applied without needing to restart xapmcsRain8Net.

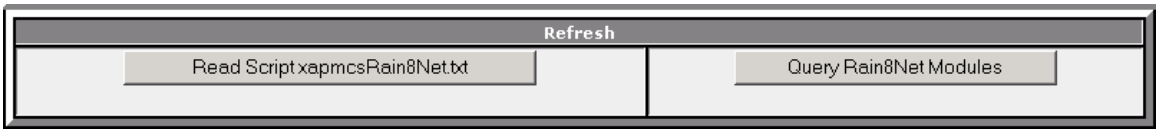


Figure 7 Integration Support Buttons

5 Scripting Interface

A Tag is a user-defined name that is available from the scripting interface. The contents of the Tag name will be the current status of the component. This status can be obtained from script with the object’s Request method.

For the Rain8Net values the name of the Tag, and one of the two special words “Value” or “LastChange” are used. For example:

```
Status = xap.Request(“Zone1”)("value")
LastChange = xap.Request(“Zone2”)("LastChange")
```

For xAP received messages the Key from the message body is used rather than “Value”. “LastChange” is also available and will return the time the messsage was received.
For control of the valves from script using the object’s Response method such as the following to set the tag item “Zone3” to the “Off” state::

```
xap.Response "Zone3", "off"
```

All scripting performed locally under activation by xapmcsRain8Net is routed through xapmcsRain8Net.txt which is located in the \Scripts folder under xapmcsRain8Net.exe. This file is called periodically and on event based upon configuration settings in Section 4.

This local scripting is independent from other scripting such is available via Homeseer. It allows local control decisions to be made based upon Rain8Net information and information available from xAP messages.

5.1 Methods

5.1.1 xAP Message Related Methods

Public Sub Response(sTag As Variant, sValue As Variant)
    Command a Rain8Net feature or xAP message
Public Function Request(sTag As Variant) As Variant
    Get value from a key within a received message or Rain8Net valve
Public Function TagList(sTag As Variant) As Variant
    Obtain collection of Tags
Public Function Sequence(sTag As Variant) As Variant
    Get relative number of last change of Rain8Net valve or xAP Message
Public Sub X10(sDevice As Variant, sCommand As Variant, Optional sBrightness As Variant = 0)
    Send command using xap-x10 schema
Public Sub SendXapMessage(sTarget As Variant, sClass As Variant, sSection As Variant, sData As Variant, sSubaddress As Variant, sUIDIn As Variant, sTargetSubaddress As Variant)
    Send a general case xAP message

5.1.2 General Scripting Support

Public Sub Run(sFilename As Variant)
Public Function RunEx(sFilename As Variant, sProcedure As Variant, sParameter As Variant) As Variant
Public Function Spawn(sScriptName As Variant, sProcedure As Variant, sScriptParm As Variant) As Object 'WshExec
Public Function Sunset() As Variant
Public Function Sunrise() As Variant
Public Property Get GetAppPath() As String
Public Sub WriteLog(c As Variant, message As Variant)

5.1.3 Ini File Support

Public Sub SaveIniSetting(Group As String, KEY As String, Value As String, Optional sFile As String)
Public Function GetIniSetting(Group As String, KEY As String, Default As String, Optional sFile As String) As String
Public Function GetINISection(Group As String, Optional sFile As String) As String
Public Sub ClearINISection(Group As String, sFile As String)
Public Sub FlushIniCache()

5.2 Script Example – \Scripts\xapmcsRain8Net.txt

```
sub main(sTag)

	'================== Script Objective ============================
```

```
	'Turn valve 1,2,3 off whenever daily rainfall > 0
	'Value 1 is Tagged as Zone1
	'Value 2 is Tagged as Zone2
	'Valve 3 is Tagged as Zone3
	'Weather.Report message is Tagged as Weather
	'=================================================================

	'xap.writelog "Rain8Net Script",cstr(sTag)

	'When sTag is null it means that this is periodic call to the script
	'rather than on a data change event.  For this script there is nothing
	'to do on periodic called if it occurs

	if sTag = "" then
		exit sub
	end if


	'==============================
	'Weather Report Processing
	'==============================
	if sTag = "Weather" then

		'native properties for internal items are "value" and "lastchange"
		'properties for received xAP messages will be a function of the schema

		'status =  xap.Request("Zone1")("value")      'internal valve example to
get current valve status
		'changed = xap.Request("Zone2")("lastchange") 'internal valve example to
get time of last change
		'xap.writelog "Status/Changed", cstr(status) & "," & cstr(changed)

		rainfall = xap.Request("Weather")("weather.report.raini") 'get rainfall
from received message
		'xap.writelog "Rain",cstr(rainfall)

		if rainfall > 0 then
			xap.Response "Zone1", "OFF"
			xap.Response "Zone2", "OFF"
			xap.Response "Zone3", "OFF"
		end if
	End if

	'================ commentary on debug and getting message data ==========

	'write to the log for debug
	'xap.writelog "STAG",sTag

	'Get the message info that caused this script to be called.

	'When using X10 or other schema that does not resolve to a unique address
	'in the header such as xapbsc does then "LastEvent" should be used
	'If "LastEvent" is not used then the values returned from the requested
	'collection will be the values from the last message requested for the
	'tagged item and it may not be the values from the message that generated
	'the event call.  e.g. F4 message follwed by F3 message will generate
	'and F4 event, but the F3 data may now be reflected in the current X10 tag
	'=====================================================================


	'=====================================================================
	'Reference Example processing of an xAPBSC message using the State key
	'=====================================================================
	'if sTag = "Fireplace" then
	'	Value = ucase(xap.Request("Fireplace")("input.state.1.state"))
	'	if Value = "ON" then
	'		xap.Response "DenonVolume", UP
	'	Else
	'		xap.Response "DenonVolume", DOWN
	'	End if
	'	exit sub
```

```
        'End if

End Sub
```