



**RSView SE 4.0**

## **VBA Error Handling For Communication Events**

**Tips & Best  
Practices**

**Version 1.0**





## Table of Contents

Introduction .....	3
Tag Object communication error handling fundamentals .....	3
Basic error handling responses .....	4
Reading a tag value and responding to returned errors .....	4
Writing a tag value and responding to returned errors .....	8
Managing tag status changes by using a Group_Change event .....	11
Using RefreshFromSource to verify Tag communication .....	16
Using RefreshFromSource to verify Tag Group communication .....	19
Using cascading WritePendingValues to perform retries .....	21
Examples of a retry mechanism .....	25
Summary .....	25
Sample 'Retry' functions in RETRIES.BAS module .....	27
TagGroupAddWithRetries .....	27
TagReadWithRetries .....	29
TagWriteWithRetries .....	31
TagRFSWithRetries .....	33
TagGroupWPVWithRetries .....	35
Error codes to consider .....	37
Tag Quality constants .....	39
Tag Quality SubStatus constants .....	40
References .....	42
Appendix A <i>RETRIES.BAS</i> VBA Module .....	43



## Introduction

This document has been created as a repository for best practices and tips focused on recognizing and responding to communication errors when using tags in VBA within RSVIEW SE. Specifically it covers how to recognize potential situations where communication events can affect intended actions in VBA, as well as how to arrange code to allow alternate code paths should an action or item be unavailable due to same.

This document also discusses an example of one of many potential methods for implementing retries of Tag and Tag Group actions should a communication error during operation. Through methods like this multiple attempts of an action can be made before determining if an alternate code path is required for an error condition.

This document does not cover specific responses to these conditions. Each end application can vary dramatically in what it would consider an appropriate response. Methods and sample code provided must be reviewed and tested to determine appropriate suitability.

This document is intended to be a living document; to be updated with suggestions and examples recognized to have a best practices element to be shared with others.

## Tag Object communication error handling fundamentals

When developing an application's VBA code for use in RSVIEW SE graphic displays, it is important to consider and prepare to respond to any disruptions that could occur while using Tag and Tag Group objects. The setting of a value, reading a current value, adding a tag to a group and other operations by themselves do not guarantee success. Communication bottlenecks, offline controllers, busy servers, user rights, or other influences can result in errors through these operations. At minimum when coding it should be considered what alternate code path should be executed should the Tag or Tag Group action fail at any instance it is performed. Not handling these situations can also result in unpredictable operation. If no or incorrect error is used, possible lockup of the VBA engine on the client can occur in some circumstances.

While "an appropriate response or code path" to an incident can vary from installation to installation, the tools available to make the decisions are all the same. The state of the overall running system, VBA's asserted Err.Number, returned Quality codes, Tag last error codes, and string lists of affected tag names can all be used as advisory indicators to determine an appropriate response, as well to help decide if subsequent attempts should be made.



## Basic error handling responses

The following are examples of various tag and tag group actions. Each is shown with a basic code example showing how its error conditions can be evaluated.

### *Reading a tag value and responding to returned errors*

This example shows a basic method of monitoring for error conditions while configuring a tag group and reading a tag. The first subroutine creates a TagGroup object on the loading of a display, adds tags to it, and activates the Tag Group. The second subroutine reads the value of a tag in the Tag Group. The third performs some housekeeping on the closing of the display.

Line #	Code
1	Option Explicit
2	Dim WithEvents oGroup As TagGroup
3	
4	
5	Sub Display_Animationstart()
6	
7	On Error Resume Next
8	
9	Err.Clear
10	
11	If oGroup Is Nothing Then
12	
13	Set oGroup = Application.CreateTagGroup(Me.AreaName, 500)
14	
15	'Since we are using 'On Error Resume Next' need to check if an
16	'error was triggered in creating the group. If so, record and exit.
17	If Err.Number Then
18	
19	LogDiagnosticsMessage "Error creating TagGroup. Error: " _
20	& Err.Description, ftDiagSeverityError
21	
22	Exit Sub
23	
24	End If



Line #	Code
25	
26	oGroup.Add "System\Second"
27	
28	oGroup.Add "AnalogTag1"
29	
30	oGroup.Active = True      'Activate the tag group
31	
32	End If
33	
34	End Sub
35	
36	
37	
38	Sub GetTagValue()
39	'This procedure can be called by an event or other procedure
40	
41	On Error Resume Next
42	
43	Dim oTag As Tag
44	
45	Dim vValue As Variant
46	
47	'Need to confirm the tag group exists before trying to use it
48	If Not oGroup Is Nothing Then
49	
50	Set oTag = oGroup.Item("AnalogTag1") 'or your tag
51	
52	'Make sure its an acceptable tag
53	
54	If Err.Value = 0 Then
55	
56	Err.Clear
57	
58	vValue = oTag.Value
59	
60	
61	'In this example we only show 2 error state traps. In a section
62	'further in this document, POSSIBLE ERROR CODES TO
63	'CONSIDER, you will find other error conditions you may want to
64	'respond with specific actions in their own 'Case' statements
65	Select Case Err.Number
66	
67	Case 0:
68	'In reality we would use the LogDiagnostics Message
69	'to report messages rather than a MsgBox as all VBA code is
70	'halted when a message box appears. For this example we



Line #	Code
71	'will just use a MsgBox for simplicity.
72	MsgBox "AnalogTag1 = " & vValue
73	
74	
75	
76	Case tagErrorReadValue:
77	'In some cases, this can be as simple as the initial value
78	'has not been retrieved from the device yet (too soon).
79	'You may instead want to consider a short delay and a retry.
80	MsgBox "Error to reading tag value. Error: " & _
81	oTag.LastErrorString
82	
83	
84	
85	Case tagErrorOperationFailed:
86	
87	MsgBox "Failed to read from tag. Error: " & Err.Description
88	
89	
90	Case Else
91	
92	'at minimum, report the error
93	LogDiagnosticsMessage "Error occurred!" & _
94	"Error# 0x" & Hex(Err.Number) & ", " & Err.Description, _
95	ftDiagSeverityError
96	
97	
98	End Select
99	
100	Else
101	
102	'If the tag could not be referenced ( ie. it isn't in the tag group, this code will execute)
103	LogDiagnosticsMessage "Error occurred in GetTagValue. Could not assign tag from tag
104	group", _
105	ftDiagSeverityError
106	
107	
108	End If
109	
110	
111	Else
112	'If the tag group doesn't exist an appropriate response for your
113	'situation should occur here
114	LogDiagnosticsMessage "Error occurred in GetTagValue. Tag Group does not exist", _
115	ftDiagSeverityError
116	



Line #	Code
117	End If
118	
119	Set oTag = Nothing
120	
121	
122	End Sub
123	
124	Sub Display_BeforeAnimationStop()
125	
126	'housekeeping
127	Set oGroup = Nothing
128	
129	End Sub



## Writing a tag value and responding to returned errors

This example shows a basic method of monitoring for error conditions while configuring a tag group and after writing a single tag value. The first subroutine creates a TagGroup object on the loading of a display, adds tags to it, and activates the Tag Group. The second subroutine identifies a tag in the group to write, sets the value, and evaluates any error conditions returned. The third performs some housekeeping on the closing of the display

Note, the tag group does not need to be active in order to perform tag writes. This is only shown to be consistent with the other examples in this document.

Line #	Code
1	Option Explicit
2	
3	Dim WithEvents oGroup As TagGroup
4	
5	
6	Sub Display_AnimationStart()
7	
8	On Error Resume Next
9	
10	Err.Clear
11	
12	If oGroup Is Nothing Then
13	
14	Set oGroup = Application.CreateTagGroup(Me.AreaName, 500)
15	
16	If Err.Number Then
17	
18	LogDiagnosticsMessage "Error creating TagGroup. Error: " _
19	& Err.Description, ftDiagSeverityError
20	
21	Exit Sub
22	
23	End If
24	
25	oGroup.Add "AnalogTag1"
26	
27	'If you have other tags that you would be using in this display's code
28	' and require them at the same update rate as the tag above, add them
29	' in with additional oGroup.Add methods
30	' ie. oGroup.Add "AnalogTag2"...





78	MsgBox "Unable to write tag value. Client is read-only."
79	
80	
81	Case tagErrorWriteValue:
82	
83	If oTag.LastErrorNumber = tagErrorInvalidSecurity Then
84	'In this example we are using a MsgBox, however in
85	'reality you should log this and/or respond appropriately
86	'with your application's needs
87	MsgBox "Unable to write tag value. The current user " & _
88	"does not have security rights."
89	
90	Else
91	'In this example we are using a MsgBox, however in
92	'reality you should log this and/or respond appropriately
93	'with your application's needs
94	MsgBox "Error writing tag value. Error: " & _
95	oTag.LastErrorString
96	
97	End If
98	
99	
100	
101	Case tagErrorOperationFailed:
102	
103	'In this example we are using a MsgBox, however in
104	'reality you should log this and/or respond appropriately
105	'with your application's needs
106	MsgBox "Failed to write to tag. Error: " & Err.Description
107	
108	
109	End Select
110	
111	End If
112	
113	
114	End Sub
115	
116	Sub Display_BeforeAnimationStop ()
117	'housekeeping
	Set oGroup = Nothing
	End Sub



## Managing tag status changes by using a Group\_Change event

One common method of detecting and managing a communication disruption is through the use of a tag group event handler. An event handler can be established to start a procedure should the value or status of a tag in a group change.

Through running the procedure on this event for an active group of tags on scan it can be determined if good values cannot be collected and allow for appropriate response relevant to the individual application. The configuration and procedure for this event handler can be established by running code similar to the example below. It can be hosted in either a display hidden in the background and always updating (“/ZA” parameter), or as part of a display that is always shown on the screen such as on a menu or alarm bar graphic.

Note: Try to keep the number tags in the group to a minimum to achieve the monitoring you desire. A change in value or state for any of the tags in the group will trigger the group’s event procedure. If your application is very VBA intensive, excessive processing of this code could have an impact on the responsiveness of your other VBA code as only one procedure can run at any given time.

Line#	Code
1	Option Explicit
2	Dim WithEvents oGroup as TagGroup
3	
4	Sub Display_AnimationStart()
5	'This routine sets up the tag group upon loading the display
6	
7	On Error Resume Next
8	
9	Err.Clear
10	
11	If oGroup Is Nothing Then
12	
13	'This will set the group to be monitored every 500mSec. If slower
14	'monitoring is acceptable, increase this 2nd parameter to reduce load
15	
16	Set oGroup = Application.CreateTagGroup(Me.AreaName, 500)
17	
18	If Err.Number Then
19	
20	LogDiagnosticsMessage "Error creating TagGroup. Error: " _
21	& Err.Description, ftDiagSeverityError
22	
23	Exit Sub
24	
25	End If



Line#	Code
26	
27	oGroup.Add "AnalogTag1" ' Or name of applicable device tag
28	' If monitoring additional tags are desired, put here
29	oGroup.Active = True
30	
31	End If
32	
33	End Sub
34	
35	
36	Private Sub oGroup_Change(ByVal TagNames As StringList)
37	'This is the procedure that is triggered by a change in tag values or quality
38	
39	Dim vTagName As Variant
40	Dim vValue As Variant
41	Dim vTimeStamp As Date
42	Dim vQuality As tagQualityConstants
43	Dim vSubStatus As tagSubStatusConstants
44	Dim vLimit As tagLimitConstants
45	Dim vShouldResume As Boolean
46	Dim oTag As Tag
47	
48	On Error GoTo ErrHandler:
49	
50	'The Change() event can occur if the tag Value or Quality changes. In
51	'this case we are checking the quality of the tags in the group
52	
53	
54	'Go through the list of provided tag names. Each of these had its
55	'quality or value changed, or both
56	For Each vTagName In TagNames
57	
58	Set oTag = oGroup.Item(vTagName)
59	oTag.GetTagData vValue, vTimeStamp, vQuality, vSubStatus, vLimit
60	
61	Debug.Print vTagName & ";" & Now
62	
63	'If the tag is reporting back as an empty value, it is considered bad. In
64	'this case an ERR value should have triggered with the GetTagData
65	'method just before this. Another way to do this check is to compare
66	'vQuality with tagQuality
67	If VarType(vValue) = vbEmpty Then
68	
69	Call oTagStateBad(vTagName, vSubStatus, oTag.LastErrorString)
70	
71	ElseIf vQuality = tagQualityUncertain Then



Line#	Code
72	Call oTagStateStale(vTagName, vSubStatus, vTimeStamp)
73	
74	
75	End If
76	
77	Next
78	
79	
80	Set oTag = Nothing
81	
82	Exit Sub
83	
84	
85	ErrorHandler:
86	
87	vShouldResume = False
88	
89	' Using Select Case,
90	Select Case Err.Number
91	
92	Case tagErrorReadValue:
93	'Tried to read a bad Tag value
94	'in this example case the scope is limited. Let the code continue
95	vShouldResume = True
96	
97	Case tagErrorOperationFailed:
98	'Something didn't work in an overall general fashion - as in???
99	' Respond accordingly
100	Case Else
101	'at minimum, report the error
102	LogDiagnosticsMessage "Error occurred in oGroupChange ()." & _
103	"Error# 0x" & Hex(Err.Number) & ", " & Err.Description, _
104	ftDiagSeverityError
105	
106	End Select
107	
108	If vShouldResume = True Then
109	Err.Clear
110	Resume Next
111	Else
112	'Housekeeping
113	Set oTag = Nothing
114	End If
115	End Sub
116	
117	



Line#	Code
118	
119	Sub oTagStateStale(ByVal vWhat As Variant, ByVal vWhy As _
120	tagSubStatusConstants, vTimeStamp As Date)
121	'This procedure manages a response to a tag reported as having a
122	'Stale quality
123	
124	On Error GoTo ErrHandler
125	
126	'Values can go temporarily stale for various reasons due to bandwidth
127	'transitory communication problems and such. If desired to monitor, here
128	'is where you would react to a stale tag. In this case we will just log the
129	'information. In reality these could be nuisance messages.
130	
131	LogDiagnosticsMessage "Tag " & vWhat & " has been reported stale." & _
132	"SubStatus code (" & vWhy & "), Last Reported ." & vTimeStamp,
133	ftDiagSeverityWarning
134	
135	'At this point add in whatever code is relevant to your application
136	'Trigger a special alarm, retry, condition based specific tag name, etc.
137	
138	Exit Sub
139	
140	ErrHandler:
141	
142	LogDiagnosticsMessage "Error occurred in oTagStateStale()." & _
143	"Error# 0x" & Hex(Err.Number) & ", " & Err.Description, _
144	ftDiagSeverityError
145	End Sub
146	
147	Sub oTagStateBad(ByVal vWhat As Variant, ByVal vWhy As _
148	tagSubStatusConstants, vReported As Variant)
149	'This procedure manages a response to a tag reported as having a
150	'bad quality
151	On Error GoTo ErrHandler
152	
153	'Generally a good idea to report it, unless it is an occasionally expected
154	'error, in which case logging it may make this a nuisance.
155	
156	LogDiagnosticsMessage "Unable to read value of tag " & _
157	vWhat & ". The quality of the value is Bad. Last Reported " & _
158	"Error:" & vReported, ftDiagSeverityWarning
159	
160	'At this point add in whatever code is relevant to your application
161	'Trigger a special alarm, retry, condition based specific tag name, etc.
162	
163	Exit Sub



Line#	Code
164	
165	ErrorHandler:
166	
167	LogDiagnosticsMessage "Error occurred in oTagStateBad()." & _
168	"Error# 0x" & Hex(Err.Number) & ", " & Err.Description, _
169	ftDiagSeverityError
170	End Sub
171	
172	Sub Display_BeforeAnimationStop()
173	
174	'Housekeeping
175	Set oGroup = Nothing
176	
177	End Sub





```
36      End If
37
38      oGroup.Add "AnalogTag1" ' If monitoring additional tags are desired
39      oGroup.Item(oGroup.Count).RefreshFromSource 'Pull a current value. If there is a communications
40          'error, the ERR value will be set
41      If Err.Number Then
42          Call ErrHandler(oGroup.Item(oGroup.Count).Name)
43
44      End If
45
46      oGroup.Active = True
47
48      MsgBox "Complete!"
49
50      End If
51
52      End Sub
53
54      Sub ErrHandler(ByVal vName As String)
55          'This shows only a couple of examples of the considered states that can be evaluated. Also in a real
56          'application the use of MsgBox is not recommended as it halts all VBA code until it is closed.
57
58          Select Case Err.Number
59
60              Case tagErrorReadValue:
61                  'Tried to read a bad Tag value
62                  MsgBox " Tag " & vName & " was marked as Bad"
63
64              Case tagErrorOperationFailed:
65                  'Something didn't work. Respond in a general fashion
66                  MsgBox " Tag " & vName & " did not succeed. The operation failed"
67
68              Case Else
69                  'at minimum, report the error
70                  LogDiagnosticsMessage "Error occurred in Display_AnimationStart(). Tag Name: " & _
71                      vName & ", Error# 0x" & Hex(Err.Number) & ", " & Err.Description, _
72                      ftDiagSeverityError
73
74          End Select
75
76          Err.Clear
77
78      End Sub
79
80
81
82
```



83	Sub Display_BeforeAnimationStop()
84	
85	'Housekeeping
86	Set oGroup = Nothing
87	
	End Sub



## Using RefreshFromSource to verify Tag Group communication

In a similar manner to the prior example, communication errors for tags can optionally be checked as a collection by using the *RefreshFromSource* method on the Tag group rather on individual tags. This solution has the advantage of providing a list of the tags in error after attempting to put the whole group in scan. However, if it is desired to react after the first one or two tags in error, the prior method should be used as the example below waits until all tags have been attempted to be read before resuming code operation. When there are a larger number of tags in error this method could take longer to respond as it waits for all timeouts.

Line#	Code
1	Option Explicit
2	Dim WithEvents oGroup As TagGroup
3	
4	
5	Sub Display_AnimationStart()
6	'This routine sets up the tag group upon loading the display
7	
8	Dim vListTagsInError As StringList
9	Dim RefreshSuccess As Boolean
10	
11	
12	On Error Resume Next
13	
14	Err.Clear
15	
16	If oGroup Is Nothing Then
17	
18	'This will set the group to be monitored every 500mSec. If slower
19	'monitoring is acceptable, increase this 2nd parameter to reduce load
20	
21	Set oGroup = Application.CreateTagGroup(Me.AreaName, 500)
22	
23	If Err.Number Then
24	
25	LogDiagnosticsMessage "Error creating TagGroup. Error: " _
26	& Err.Description, ftDiagSeverityError
27	
28	Exit Sub
29	
30	End If
31	
32	'Note the grouping here. After adding an item we attempt to pull the current value for that tag



33	'then check to see if it generated an error, launching into an error handler if it did
34	
35	On Error GoTo ErrHandler
36	
37	oGroup.Add "AnalogTag1" ' Or name of device tag
38	oGroup.Add "AnalogTag2" 'If monitoring additional tags are desired
39	
40	oGroup.Active = True
41	
42	RefreshSuccess = oGroup.RefreshFromSource(vListTagsInError) 'Note this will pause VBA
43	execution until all tags in the
44	'group have been requested.
45	
46	If RefreshSuccess = False Then
47	
48	'The appropriate response logic would be here. This example only identifies an error occurred
49	LogDiagnosticsMessage "TagGroup RefreshFromSource failed", ftDiagSeverityError
50	
51	End If
52	
53	End If
54	
55	Exit Sub
56	
57	
58	ErrHandler:
59	
60	LogDiagnosticsMessage "Error occurred in Display_AnimationStart()." & _
61	", Error# 0x" & Hex(Err.Number) & ", " & Err.Description, _
62	ftDiagSeverityError
63	
64	
65	End Sub
66	
67	
68	
69	Sub Display_BeforeAnimationStop()
70	
71	'Housekeeping
72	Set oGroup = Nothing
73	
74	End Sub



## Using cascading WritePendingValues to perform retries

*WritePendingValues* allows for a set of tag objects to be set with pending write values and triggered as a collective set of writes when all have been loaded with values. This method also provides the ability to retry writes that were unsuccessful. When a *WritePendingValues* method has completed, the tags which writes were unsuccessful still retain their *PendingWriteValues*, allowing for a second *WritePendingValues* to occur with only attempts to write the previously unsuccessful ones.

1	Option Explicit
2	Dim WithEvents oGroup As TagGroup
3	
4	
5	
6	Sub Display_AnimationStart()
7	'This routine sets up the tag group upon loading the display
8	
9	On Error Resume Next
10	
11	Err.Clear
12	
13	If oGroup Is Nothing Then
14	
15	'This will set the group to be monitored every 500mSec. If slower
16	'monitoring is acceptable, increase this 2nd parameter to reduce load
17	
18	Set oGroup = Application.CreateTagGroup(Me.AreaName, 500)
19	
20	If Err.Number Then
21	
22	LogDiagnosticsMessage "Error creating TagGroup. Error: " _
23	& Err.Description, ftDiagSeverityError
24	
25	Exit Sub
26	
27	End If
28	
29	'Note the grouping here here. After adding an item we attempt to pull the current value for that tag
30	'then check to see if it generated an error, launching into an error handler if it did
31	
32	oGroup.Add "AnalogTag1" ' Or name of device tag
33	
34	
35	oGroup.Item(oGroup.Count).RefreshFromSource 'Pull a current value. If there is a communications
36	'error, the ERR value will be set
37	If Err.Number Then



```
38         Call ErrHandler(oGroup.Item(oGroup.Count).Name)
39
40     End If
41
42     oGroup.Add "AnalogTag2" ' If monitoring additional tags are desired
43     oGroup.Item(oGroup.Count).RefreshFromSource 'Pull a current value. If there is a communications
44         'error, the ERR value will be set
45     If Err.Number Then
46         Call ErrHandler(oGroup.Item(oGroup.Count).Name)
47
48     End If
49
50     oGroup.Active = True
51
52 End If
53
54 End Sub
55
56 Private Sub WriteValues()
57
58     On Error Resume Next
59
60     Dim oTag As Tag
61     Dim WriteSuccess As Boolean
62     Dim Missedtags As StringList
63
64
65     'Need to confirm the tag group exists before trying to use it
66     If Not oGroup Is Nothing Then
67
68         Set oTag = oGroup.Item("AnalogTag1") 'or your tag
69
70         Err.Clear
71
72         oTag.PendingWriteValue = 10 ' Or your given value
73
74
75         If Err.Number = 0 then
76
77
78
79             WriteSuccess = oGroup.WritePendingValues(Missedtags)
80
81             If WriteSuccess = True Then
82
83                 LogDiagnosticsMessage "AnalogTag1= " & oTag.Value, ftDiagSeverityInfo
84
```



85	Else
86	
87	WriteSuccess = oGroup.WritePendingValues(Missedtags)
88	
89	If WriteSuccess = True Then
90	
91	LogDiagnosticsMessage "After Retry, AnalogTag1= " & oTag.Value,
92	ftDiagSeverityInfo
93	
94	Else
95	
96	LogDiagnosticsMessage "Write Failed. respond accordingly ",
97	ftDiagSeverityError
98	
99	End If
100	End If
101	
102	
103	
104	
105	
106	Else
107	
108	'at minimum, report the error
109	LogDiagnosticsMessage "Error occurred!" & _
110	"Error# 0x" & Hex(Err.Number) & ", " & Err.Description, _
111	ftDiagSeverityError
112	
113	
114	End If
115	
116	
117	
118	Else
119	'If the tag group doesn't exist an appropriate response for your
120	'situation should occur here.
121	LogDiagnosticsMessage "Error occurred in WriteValues. Tag Group does not exist", _
122	ftDiagSeverityError
123	
124	End If
125	
126	
123	End Sub
124	
125	
126	
127	



128	Sub ErrHandler(ByVal vName As String)
129	'This shows only a couple of examples of the considered states that can be evaluated. Also in a real
130	'application the use of MsgBox is not recommended as it halts all VBA code until it is closed.
131	
132	Select Case Err.Number
133	
134	Case tagErrorReadValue:
135	'Tried to read a bad Tag value
136	LogDiagnosticsMessage " Tag " & vName & " was marked as Bad", ftDiagSeverityError
137	
138	Case tagErrorOperationFailed:
139	'Something didn't work. Respond in a general fashion
140	LogDiagnosticsMessage " Tag " & vName & " did not succeed. The operation failed",
141	ftDiagSeverityError
142	
143	Case Else
144	'at minimum, report the error
145	LogDiagnosticsMessage "Error occurred in Display_AnimationStart(). Tag Name: " & _
146	vName & ", Error# 0x" & Hex(Err.Number) & ", " & Err.Description, _
147	ftDiagSeverityError
148	
149	End Select
150	
151	
152	Err.Clear
153	
154	End Sub
155	
156	
157	Sub Display_BeforeAnimationStop()
158	
159	'Housekeeping
160	Set oGroup = Nothing
161	
162	End Sub
163	

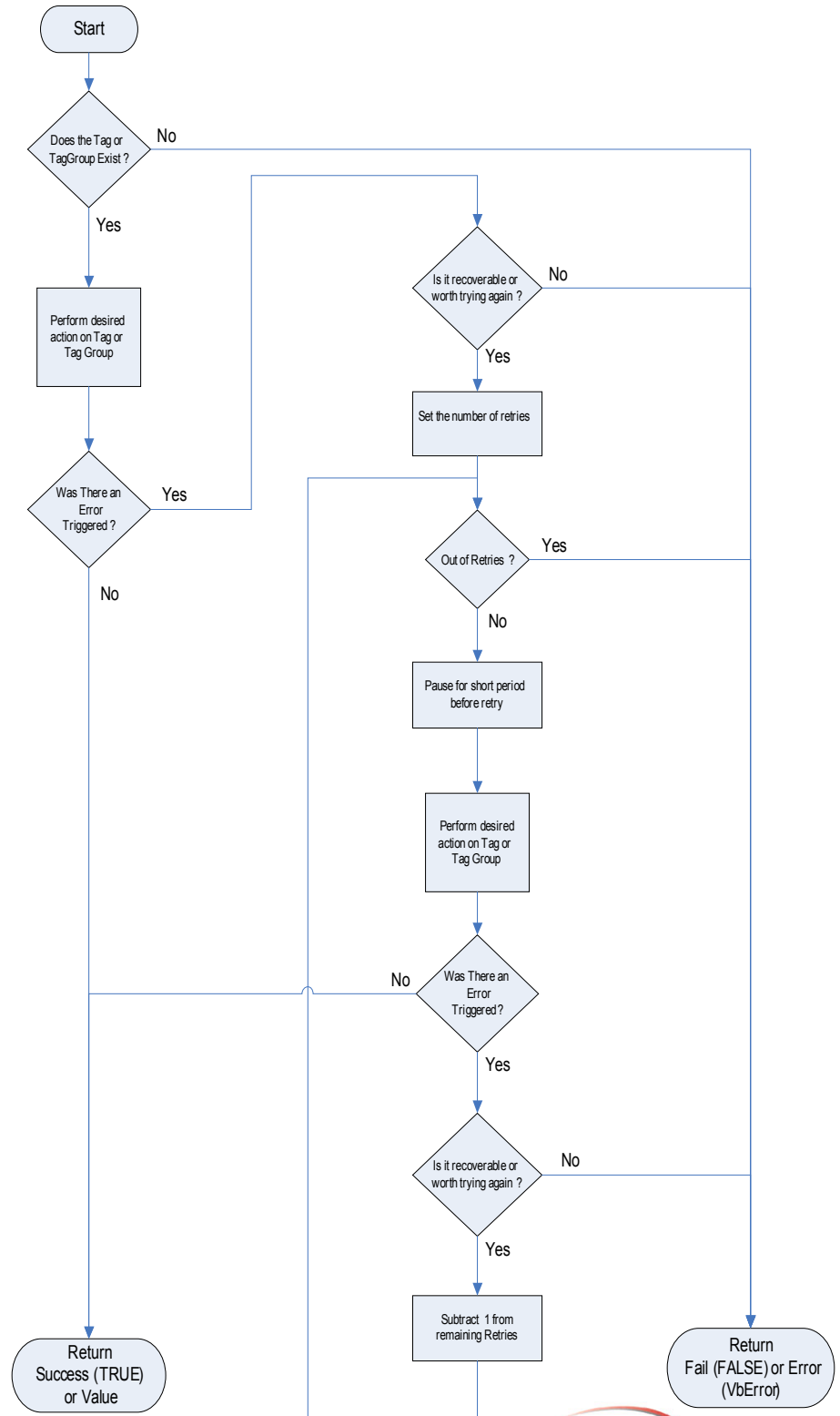


## Examples of a retry mechanism

### Summary

There are books filled with different strategies on how retries can be managed and this document does not attempt to cover them all. However, this is an example flow chart using one general method of performing retries.

This flowchart is based on just determining at a high level if the action was successful or not. From implementation at run time, it can be determined if an alternate code path is required to react to an attempted action that failed.





Example code has also been created based on this flow chart for various common Tag / Tag Group activities and is contained in the module RETRIES.BAS available with this document. It is also listed in Appendix A for review.

RETRIES.BAS can be added as a VBA module to a given graphic display for use by code within that display. It may or may not fit your particular environment or requirements. You must ensure any code added to a production system must be fully tested before introduction to ensure it meets your specific needs.

In RETRIES.BAS are examples of some potential wrapping functions for common activities - a Tag Write, a Tag Read, Tag Group Add, Tag Group Write Pending Value, and Tag RefreshFromSource. These functions allow for a configurable number of retries and retry delays for a given attempt to perform them.

**CAUTION:** Using this method of a retry mechanism will hold off other VBA code while these functions attempt their retries. In addition events will be queued during this time. Use retry numbers and delays settings that are practical and consider overall operation needs. In addition, these routines do not pass back why a failure may have occurred, they only identify if it was successful or not so that an alternate code path can be implemented if the attempt did fail.

**NOTE:** All routines use a common *IsCommErrorRecoverable* function which is also included as part of Appendix A. This function is used to record and an error event in the diagnostics log and to determine if a particular error should be considered worth retrying or not. The classification used may need to also be modified to suit your particular production environment needs.



## Sample 'Retry' functions in RETRIES.BAS module

The following are descriptions of the sample functions provided in RETRIES.BAS. They based on the retry logic provided in the flowchart in the previous section, serving as wrappers around existing functionality.

The example usage code assumes a tag group oGroup has been created and contains tags AnalogTag1 and AnalogTag2.

### TagGroupAddWithRetries

This function attempts to add a Tag to a Tag Group, using retries if needed. If successful, it returns TRUE, if not it returns FALSE.

The syntax for the **TagGroupAddWithRetries** function is:

**TagGroupAddWithRetries** (**TagGroup**, **TagName**, *ReferenceString* , *RetryCount*, *RetryDelay* )

Where:

**TagGroup** is a TagGroup object that will have the tag attempted to be added it.

**TagName** is a string containing the name of the tag to be added to the group **TagGroup**.

**ReferenceString** is an optional string parameter which can be used to show specific tracing text in error messages logged for failed attempts in order to aid troubleshooting.

**RetryCount** is an optional long integer parameter to specify the maximum number of retries that should be used before considering this a fail, should the initial attempt fail. If not specified, the function will use the value in the constant **TagRetriesDefaultRetries**. In the example code as provided, it is set to 3 retries.

**RetryDelay** is an optional long integer parameter to specify how long in milliseconds should the function wait between retry attempts. If not specified, the system will use the value in the constant **TagRetriesDefaultRetryDelay**. In the example code as provided, it is set to 100 milliseconds.



### TagGroupAddWithRetries Example Usage:

```
1 Private Sub Button6_Released()  
2  
3 If Not oGroup Is Nothing Then  
4     If TagGroupAddWithRetries(oGroup, "AnalogTag3") = False Then  
5         MsgBox "not added. Respond accordingly"  
6     Else  
7         MsgBox "added"  
8     End If  
9 End If  
10  
11 Exit Sub  
12  
13 ErrHandler:  
14     LogDiagnosticsMessage Err.Description & " ( Error# 0x" & Hex(Err.Number) & _  
15     " in Display "" & Me.FullName & "" - Button6_Released )", ftDiagSeverityError  
16  
17 End Sub
```



## TagReadWithRetries

This function attempts to read a value from a tag, using retries. If successful, it returns the value of the tag; if not successful it returns **vbError**.

The syntax for the **TagReadWithRetries** function is:

**TagReadWithRetries** (*oTag*, *ReferenceString* , *RetryCount*, *RetryDelay* )

Where:

**oTag** is a Tag object that references the tag to be read.

**ReferenceString** is an optional string parameter which can be used to show specific tracing text in error messages logged for failed attempts in order to aid troubleshooting.

**RetryCount** is an optional long integer parameter to specify the maximum number of retries that should be used before considering this a fail, should the initial attempt fail. If not specified, the function will use the value in the constant **TagRetriesDefaultRetries**. In the example code as provided, it is set to 3 retries.

**RetryDelay** is an optional long integer parameter to specify how long in milliseconds should the function wait between retry attempts. If not specified, the system will use the value in the constant **TagRetriesDefaultRetryDelay**. In the example code as provided, it is set to 100 milliseconds.



### TagReadWithRetries Example Usage:

1	Private Sub Button2_Released()
2	'Tag Read
3	On Error GoTo ErrHandler
4	Dim dtag As Tag
5	Dim vValue As Variant
6	
7	Set dtag = oGroup.Item("AnalogTag2")
8	
9	'Testing Read
10	vValue = TagReadWithRetries(dtag, , 5) 'allow up to 5 retries
11	
12	
13	If vValue = vbError Then
14	'the read didnt happen. here's where you put your alternate logic
15	MsgBox "Read was not successful. Respond accordingly"
16	
17	Else
18	MsgBox "The value read was " & vValue
19	
20	End If
21	
22	Exit Sub
23	
24	ErrHandler:
25	LogDiagnosticsMessage Err.Description & " ( Error# 0x" & Hex(Err.Number) & " in
26	Display "" & Me.FullName & "" - Button2_Released )", _
27	ftDiagSeverityError
28	
29	
30	End Sub



## TagWriteWithRetries

This function attempts to write a value to a tag, using retries. If successful, it returns TRUE; if not successful it returns FALSE.

The syntax for the **TagWriteWithRetries** function is:

**TagWriteRetries** (oTag, *ReferenceString* , *RetryCount*, *RetryDelay* )

Where:

**oTag** is a Tag object that references the tag to have the **WriteValue** sent to it.

**WriteValue** is the value to be written to the tag.

**ReferenceString** is an optional string parameter which can be used to show specific tracing text in error messages logged for failed attempts in order to aid troubleshooting.

**RetryCount** is an optional long integer parameter to specify the maximum number of retries that should be used before considering this a fail, should the initial attempt fail. If not specified, the function will use the value in the constant **TagRetriesDefaultRetries**. In the example code as provided, it is set to 3 retries.

**RetryDelay** is an optional long integer parameter to specify how long in milliseconds should the function wait between retry attempts. If not specified, the system will use the value in the constant **TagRetriesDefaultRetryDelay**. In the example code as provided, it is set to 100 milliseconds.



### TagWriteWithRetries Example Usage:

1	Private Sub Button1_Released()
2	On Error GoTo ErrHandler
3	Dim dtag As Tag
4	
5	Set dtag = oGroup.Item("AnalogTag2")
6	
7	'Testing Write of value 20 to AnalogTag2
8	
9	If TagWriteWithRetries(dtag, 20) = True Then
10	
11	MsgBox "Tag written"
12	
13	Else
14	
15	MsgBox "tag write failed"
16	
17	End If
18	
19	Set dtag = Nothing
20	
21	Exit Sub
22	
23	ErrHandler:
24	LogDiagnosticsMessage Err.Description & " ( Error# 0x" & Hex(Err.Number) & _
25	" in Display "" & Me.FullName & "" - Button1_Released )", ftDiagSeverityError
26	
27	
28	End Sub



## TagRFSWithRetries

This function attempts to force a REFRESHFROMSOURCE for a tag in order for it to obtain an updated value direct from a device, using retries. If successful, it returns TRUE; if not successful it returns FALSE.

The syntax for the **TagRFSWithRetries** function is:

**TagRFSWithRetries (oTag, ReferenceString , RetryCount, RetryDelay )**

Where:

**oTag** is a Tag object that references the tag to have a direct read.

**ReferenceString** is an optional string parameter which can be used to show specific tracing text in error messages logged for failed attempts in order to aid troubleshooting.

**RetryCount** is an optional long integer parameter to specify the maximum number of retries that should be used before considering this a fail, should the initial attempt fail. If not specified, the function will use the value in the constant **TagRetriesDefaultRetries**. In the example code as provided, it is set to 3 retries.

**RetryDelay** is an optional long integer parameter to specify how long in milliseconds should the function wait between retry attempts. If not specified, the system will use the value in the constant **TagRetriesDefaultRetryDelay**. In the example code as provided, it is set to 100 milliseconds.



### TagRFSWithRetries Example Usage:

1	Private Sub Button4_Released()
2	'Tag RefreshFrom Source
3	
4	On Error GoTo ErrHandler
5	Dim dtag As Tag
6	
7	
8	If Not oGroup Is Nothing Then
9	
10	Set dtag = oGroup.Item("AnalogTag2")
11	
12	
13	
14	'Testing RefreshFromSource
15	If TagRFSWithRetries(dtag) = True Then
16	
17	MsgBox "Refresh Successful"
18	
19	Else
20	
21	MsgBox "Refresh did not work. respond accordingly"
22	
23	End If
24	
25	Set dtag = Nothing
26	
27	End If
28	
29	Exit Sub
30	
31	ErrHandler:
32	LogDiagnosticsMessage Err.Description & " ( Error# 0x" & Hex(Err.Number) & _
33	" in Display "" & Me.FullName & "" - Button4_Released )", ftDiagSeverityError
34	
35	
36	End Sub



## TagGroupWPVWithRetries

This function attempts to perform a WRITEPENDINGVALUES for the group specified, using retries if any individual writes are not successful in attempts. If successful for all the items with PENDING WRITEVALUES, it returns TRUE, if not it returns FALSE.

The syntax for the **TagGroupWPVWithRetries** function is:

**TagGroupWPVWithRetries** (**TagGroup**, *ReferenceString* , *RetryCount*, *RetryDelay* )

Where:

**TagGroup** is a TagGroup object that has individual tags with PENDINGWRITEVALUES already loaded.

**ReferenceString** is an optional string parameter which can be used to show specific tracing text in error messages logged for failed attempts in order to aid troubleshooting.

**RetryCount** is an optional long integer parameter to specify the maximum number of retries that should be used before considering this a fail, should the initial attempt fail. If not specified, the function will use the value in the constant TagRetriesDefaultRetries. In the example code as provided, it is set to 3 retries.

**RetryDelay** is an optional long integer parameter to specify how long in milliseconds should the function wait between retry attempts. If not specified, the system will use the value in the constant **TagRetriesDefaultRetryDelay**. In the example code as provided, it is set to 100 milliseconds.



### TagGroupWPVWithRetries Example Usage:

1	Private Sub Button3_Released()
2	'WritePendingValues
3	
4	On Error GoTo ErrHandler
5	
6	Dim bob As Variant
7	
8	If Not oGroup Is Nothing Then
9	
10	oGroup.Item("AnalogTag2").PendingWriteValue = 22
11	oGroup.Item("AnalogTag1").PendingWriteValue = 44
12	
13	'attempts to write pending values and includes a diagnostic string to be used if errors occur.
14	If TagGroupWPVWithRetries(oGroup, "My tracking string ") = True Then
15	
16	MsgBox "WritePendingValues successful"
17	
18	Else
19	
20	MsgBox "WritePendingValues failed. Respond accordingly"
21	
22	End If
23	
24	End If
25	
26	Exit Sub



## Error codes to consider

Below are several constants available that give a representation of the error codes that can be triggered and the conditions that should be considered in an error handler. The actual error numbers are shown below in Hexadecimal format. The use of the constant name in the code is encouraged for readability.

Note there are other trappable error constants for the display client object model. The following ones are commonly related to tags and tag groups.

<b>Error Constants</b>	<b>HEX(Err.number)</b>	<b>Description</b>
—	80004005	Action Failed. This is not a trappable error, but a returned error value that can be returned from an HMI or Data Server as <i>LastErrorNumber</i> . The property <i>LastErrorString</i> may provide additional details
tagErrorInvalidSecurity	80040F05	Attempt to write to an HMI tag by a user whose security level does not permit them to write to the tag
tagErrorCollectionIndex	80040F01	Collection item corresponding to index was not found
tagErrorPropertyNotSupported	80040F82	The requested tag property is not supported by the tag or the server
tagErrorOperationFailed	80040F02	The operation failed. The error message will contain a the error code and a text description
tagErrorReadOnlyAccess	80040F04	Attempt to write a tag value or values when the RSView SE Client is in Read-only mode (configured or licensed)



<b>Error Constants</b>	<b>HEX(Err.number)</b>	<b>Description</b>
tagErrorReadValue	80040F80	Unable to read a tag value because of a server error
tagErrorTagNotFound	80040F00	Unable to add the tag to the collection because the tag name was not found
tagErrorUnableToCreateGroup	80040489	The specified tag group could not be created
tagErrorUpdateRate	80040F03	Invalid (i.e., negative) update rate specified
tagErrorWriteValue	80040F81	Unable to write a tag value because of a server error



## Tag Quality constants

The along with the value of a tag, the context of the values' quality can be considered by checking against a tag object's *Quality* property. The following constants can be returned by the *Quality* property.

Constant	Quality Value	Description
tagQualityBad	0	The quality of the tag is Bad. The SubStatus property returns a value that gives additional information about why the quality is Bad.
tagQualityUncertain	1	The quality of the tag is Uncertain. The SubStatus property returns a value that gives additional information about why the quality is Uncertain.
tagQualityGood	2	The quality of the tag is Good. The SubStatus property returns a value that might give additional information about the value.



## Tag Quality SubStatus constants

Usually the Tag *Quality* status information above is sufficient to aid in deciding a course of action in VBA code. However in some cases additional information can be obtained to assist in determining an appropriate action. In some cases OPC servers can provide additional state information that may be passed through to the client VBA as SubStatus codes.

Before using *SubStatus* codes in a project it is recommended you test to verify the particular OPC server you intend to use supports the expected error code for the conditions you intend to monitor. Support for these codes vary as it is the OPC server and end equipment that return these codes to the object model.

Below is the list of constants that can be returned by the *SubStatus* property and their standard meaning:

Constant	SubStatus Value	Description
tagSSBadNonSpecific	0	The value is Bad but no specific reason is shown.
tagSSBadConfigError	1	There is some server-specific problem with the configuration. For example, the item in question has been deleted from the configuration.
tagSSBadNotConnected	2	The input is required to be logically connected to something but is not. This quality might indicate that no value is available at this time, and a possible reason could be that the value has not been provided by the data source.
tagSSBadDeviceFailure	3	A device failure has been detected.
tagSSBadSensorFailure	4	A sensor failure has been detected. The Limits property can provide additional diagnostic information in some situations.



<b>Constant</b>	<b>SubStatus Value</b>	<b>Description</b>
tagSSBadLastKnownValue	5	Communications have failed but the last known value is available.
tagSSBadCommFailuer	6	Communications have failed and there is no last known value available.
tagSSBadOutOfService	7	The block is off scan or otherwise locked. This quality is also used when the active state of the item or the group containing the item is not Active.
tagSSBadUncertainNonSpecific	8	There is no specific why the value is uncertain.
tagSSUncertainLastUsableValue	9	Whatever was writing this value has stopped doing so. The returned value should be regarded as stale. This error is associated with the failure of some external source to write the value within an acceptable period of time.
tagSSUncertainSensorCalibration	10	Either the value has been pegged at one of the sensor limits, in which case the Limit property should be set to 1 or 2, or the sensor is otherwise known to be out of calibration via some form of internal diagnostics. In this case the Limit property should be set to 0.
tagSSUncertainEngUnitsExceeded	11	The returned value is outside the limits defined for this parameter.
tagSSUncertainSubNormal	12	The value is derived from multiple sources and has less that the required number of Good sources.



<b>Constant</b>	<b>SubStatus Value</b>	<b>Description</b>
tagSSGoodNonSpecific	13	The value is good. There are no special conditions
tagSSGoodLocalOverride	14	The value has been overridden. Typically this means that the input has been disconnected and a manually-entered value has been forced

## References

- RA Technote G102842138 - Recommendations For Writing Visual Basic for Applications in RSView SE:
- <http://msdn.microsoft.com/vba/> - Microsoft's VBA homepage



## Appendix A RETRIES.BAS VBA Module

Line #	Code
1	Option Explicit
2	
3	Public Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
4	
5	Public Const TagRetriesDefaultRetryDelay = 100 ' Milliseconds
6	Public Const TagRetriesDefaultRetries = 3
7	-----
8	
9	<b>Public Function TagGroupAddWithRetries(ByRef oGroup As TagGroup, ByVal oTagName As String, Optional ReferenceString</b>
10	<b>As String = "", Optional RetryCount As Long = TagRetriesDefaultRetries, Optional RetryDelay As Long =</b>
11	<b>TagRetriesDefaultRetryDelay) As Boolean</b>
12	
13	' Module to handle adding individual tags to a tag group, as well as allowing for retries
14	' returns TRUE if successful or FALSE if not Successful.
15	
16	On Error Resume Next
17	
18	Dim RetrySuccessful As Boolean
19	Dim oTag As Tag
20	
21	Err.Clear
22	
23	TagGroupAddWithRetries = False 'Assume not Successful, unless determined otherwise
24	RetrySuccessful = False
25	
26	'Only attempt if we know we have a Tag Group
27	If oGroup Is Nothing Then
28	
29	LogDiagnosticsMessage "Unsuccessful Retry Group Add. Group Object does not exist (" & _
30	ReferenceString & ")", ftDiagSeverityError
31	
32	
33	Else



```
34 Set oTag = oGroup.Add(oTagName) ' First attempt at an add item
35
36
37 'we're also doing the extra check to confirm that a Tag object was created
38 If Err.Number = 0 And Not (oTag Is Nothing) Then
39     ' Add was Successful, return TRUE
40     TagGroup.AddWithRetries = True
41
42 Else
43
44 'Only keep trying if
45     ' 1) still have retries left,
46     ' 2) the past error was recoverable),
47     ' and 3) haven't had success yet
48     While (RetryCount > 0) And (IsCommErrorRecoverable(Err.Number, , ReferenceString & "Tag Add " & oTagName) = True)
49     And (RetrySuccessful = False)
50
51     Err.Clear
52
53
54 'Since this is a retry, take a defined break.
55 'Note that all vba will be paused during this 'Sleep'
56 'so keep the values to a minimum where possible
57 Sleep RetryDelay
58
59 Set oTag = oGroup.Add(oTagName) ' Retry attempt at adding
60
61 If Err.Number = 0 And Not (oTag Is Nothing) Then
62
63     TagGroup.AddWithRetries = True
64     RetrySuccessful = True
65
66 End If
67
68 RetryCount = RetryCount - 1
69
```



70	Wend
71	
72	'Ok , we're done with the retries.
73	'At this point we would be Successful and will return TRUE
74	'or will return FALSE. If that's the case, at least report it
75	If RetrySuccessful = False Then
76	LogDiagnosticsMessage "Unsuccessful Retry Group Add for tag '" & _
77	oTagName & "', Error# 0x" & Hex(Err.Number) & "' (" & ReferenceString & "')", ftDiagSeverityError
78	
79	End If
80	
81	End If
82	Set oTag = Nothing
83	
84	
85	End If
86	
87	Err.Clear
88	
89	
90	
91	
92	End Function
93	
94	-----
95	
96	<b>Public Function TagReadWithRetries(ByRef oTag As Tag, Optional ReferenceString As String = "", Optional RetryCount As Long</b>
97	<b>= TagRetriesDefaultRetries, Optional RetryDelay As Long = TagRetriesDefaultRetryDelay) As Variant</b>
98	
99	' Module to handle individual tag reads and allow for retries
100	' returns read value or vbError if a value could not be obtained.
101	On Error Resume Next
102	
103	Dim TempBuffer As Variant
104	Dim RetrySuccessful As Boolean
105	



106	Err.Clear
107	
108	TagReadWithRetries = vbError 'Assume not Successful, unless determined otherwise
109	RetrySuccessful = False
110	
111	'Only attempt if we know we have a Tag
112	If oTag.Is Nothing Then
113	
114	LogDiagnosticsMessage "Unsuccessful Retry Tag Read. Tag Object does not exist (" & _
115	ReferenceString & ")", ftDiagSeverityError
116	
117	Else
118	
119	TempBuffer = oTag.Value ' First attempt at a read
120	
121	If Err.Number = 0 Then
122	' read was Successful, return the read value
123	TagReadWithRetries = TempBuffer
124	
125	Else
126	
127	'Only keep trying if
128	'    1) still have retries left,
129	'    2) the past error was recoverable),
130	'    and 3) haven't had success yet
131	While (RetryCount > 0) And (IsCommErrorRecoverable(Err.Number, oTag, ReferenceString & " Tag Read") = True) And
132	(RetrySuccessful = False)
133	
134	Err.Clear
135	
136	
137	'Since this is a retry, take a defined break.
138	'Note that all vba will be paused during this 'Sleep'
139	'so keep the values to a minimum where possible
140	Sleep RetryDelay
141	



```
142 TempBuffer = oTag.Value ' Retry attempt at a read
143
144 If Err.Number = 0 Then
145     TagReadWithRetries = TempBuffer
146     RetrySuccessful = True
147
148 End If
149
150 RetryCount = RetryCount - 1
151
152 Wend
153
154 'Ok , we're done with the retries.
155 'At this point we would be Successful and will return the value
156 'or will return vbError. If that's the case, at least report it
157 If RetrySuccessful = False Then
158     LogDiagnosticsMessage "Unsuccessful Retry Read for tag '" & _
159         oTag.Name & "', Last Tag Error# 0x" & Hex(oTag.LastErrorNumber) & _
160         "'." & oTag.LastErrorString & "' (" & ReferenceString & ")", ftDiagSeverityError
161
162 End If
163
164 End If
165
166 Err.Clear
167
168 End Function
169
170 -----
171
172
173
174 Public Function TagWriteWithRetries(ByRef oTag As Tag, ByRef WriteValue As Variant, Optional ReferenceString As String =
175 "" , Optional RetryCount As Long = TagRetriesDefaultRetries, Optional RetryDelay As Long = TagRetriesDefaultRetryDelay) As
176 Boolean
177
```



178	'Module to handle individual tag Writes and allow for retries
179	'returns TRUE if successful or FALSE if not Successful.
180	On Error Resume Next
181	
182	Dim RetrySuccessful As Boolean
183	
184	Err.Clear
185	
186	TagWriteWithRetries = False 'Assume not Successful, unless determined otherwise
187	RetrySuccessful = False
188	
189	'Only attempt if we know we have a Tag
190	If oTag Is Nothing Then
191	
192	LogDiagnosticsMessage "Unsuccessful Retry Tag Write. Tag Object does not exist (" & _
193	ReferenceString & ")" , ftDiagSeverityError
194	
195	Else
196	
197	oTag.Value = WriteValue ' First attempt at a Write
198	
199	If Err.Number = 0 Then
200	' Write was Successful, return a TRUE
201	TagWriteWithRetries = True
202	
203	Else
204	
205	'Only keep trying if
206	' 1) still have retries left,
207	' 2) the past error was recoverable,
208	' and 3) haven't had success yet
209	While (RetryCount > 0) And (IsCommErrorRecoverable(Err.Number, oTag, ReferenceString & "Tag Write") = True) And
210	(RetrySuccessful = False)
211	
212	Err.Clear
213	



```
214 'Since this is a retry, take a defined break.
215 'Note that all vba will be paused during this 'Sleep'
216 'so keep the values to a minimum where possible
217 Sleep RetryDelay
218
219
220 oTag.Value = WriteValue ' Retry attempt at a Write
221
222 If Err.Number = 0 Then
223     TagWriteWithRetries = True
224     RetrySuccessful = True
225
226 End If
227
228 RetryCount = RetryCount - 1
229
230 Wend
231
232 'Ok , we're done with the retries.
233 'At this point we would be Successful and will return TRUE
234 'or will return FALSE. If that's the case, at least report it
235 If RetrySuccessful = False Then
236     LogDiagnosticsMessage "Unsuccessful Retry Write for tag '" & _
237         oTag.Name & "' with value '" & WriteValue & "', Last Tag Error# 0x" & Hex(oTag.LastErrorNumber) & _
238         ".:" & oTag.LastErrorString & "' (" & ReferenceString & ")", ftDiagSeverityError
239
240 End If
241
242 End If
243
244 End If
245
246 Err.Clear
247
248 End Function
249
```



250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285

-----

**Public Function TagRFSWithRetries(ByRef oTag As Tag, Optional ReferenceString As String = "", Optional RetryCount As Long = TagRetriesDefaultRetries, Optional RetryDelay As Long = TagRetriesDefaultRetryDelay) As Boolean**

' Module to handle individual tag RefreshFromSource and allowing for Retries

' Returns TRUE if Successful, FALSE if not successful

    On Error Resume Next

    Dim TempBuffer As Variant

    Dim RetrySuccessful As Boolean

    Err.Clear

    TagRFSWithRetries = False ' Assume not Successful, unless determined otherwise

    RetrySuccessful = False

    ' Only attempt if we know we have a Tag

    If oTag Is Nothing Then

        LogDiagnosticsMessage "Unsuccessful Retry Tag RefreshFromSource. Tag Object does not exist (" & \_  
        ReferenceString & ")", ftDiagSeverityError

    Else

        oTag.RefreshFromSource ' First attempt at a refresh

    If Err.Number = 0 Then

        ' Refresh was Successful, return True

        TagRFSWithRetries = True

    Else

        ' Only keep trying if



```
286 ' 1) still have retries left,  
287 ' 2) the past error was recoverable),  
288 ' and 3) haven't had success yet  
289 While (RetryCount > 0) And (IsCommErrorRecoverable(Err.Number, oTag, ReferenceString & "Tag RefreshFromSource") =  
290 True) And (RetrySuccessful = False)  
291  
292 Err.Clear  
293  
294  
295 'Since this is a retry, take a defined break.  
296 'Note that all vba will be paused during this 'Sleep'  
297 'so keep the values to a minimum where possible  
298 Sleep RetryDelay  
299  
300 oTag.RefreshFromSource ' Retry attempt at a refresh  
301  
302 If Err.Number = 0 Then  
303 TagRFSWithRetries = True  
304 RetrySuccessful = True  
305  
306  
307 End If  
308  
309 RetryCount = RetryCount - 1  
310  
311 Wend  
312  
313  
314 'Ok , we're done with the retries.  
315 'At this point we would be Successful and will return TRUE  
316 'or will return FALSE. If that's the case, at least report it  
317 If RetrySuccessful = False Then  
318 LogDiagnosticsMessage "Unsuccessful Retry RefreshFromSource for tag '" & _  
319 oTag.Name & "', LastTag Error# 0x" & Hex(oTag.LastErrorNumber) & _  
320 ".:" & oTag.LastErrorString & "' (" & ReferenceString & ")", ftDiagSeverityError  
321
```



322	End If
323	
324	End If
325	
326	End If
327	
328	Err.Clear
329	
330	End Function
331	
332	-----
333	
334	<b>Public Function TagGroupWPVWithRetries(ByRef oGroup As TagGroup, Optional ReferenceString As String = "", Optional</b>
335	<b>RetryCount As Long = TagRetriesDefaultRetries, Optional RetryDelay As Long = TagRetriesDefaultRetryDelay) As Boolean</b>
336	
337	'Module to handle a WritePendingValues and allow for retries
338	'on those that were Unsuccessful. Note that any Unsuccessful writes will retain their PendingWrite Value for use in
339	'retry attempts while Successful writes will not be resent.
340	,
341	'This function returns TRUE if successful or FALSE if not Successful.
342	
343	On Error Resume Next
344	
345	Dim Count As Long
346	Dim RetrySuccessful As Boolean
347	Dim MissedTags As StringList
348	Dim TestPendingWrite as Boolean
349	
350	
351	Err.Clear
352	
353	TagGroupWPVWithRetries = False 'Assume not Successful, unless determined otherwise
354	RetrySuccessful = False
355	
356	'Only attempt if we know we have a Tag Group
357	If oGroup Is Nothing Then



358	
359	LogDiagnosticsMessage "Unsuccessful Retry Group WritePendingValues. Group Object does not exist (" & _
360	ReferenceString & ")", ftDiagSeverityError
361	
362	
363	Else
364	
365	TestPendingWrite = oGroup.WritePendingValues(MissedTags) ' First attempt at a Write
366	
367	If (Err.Number = 0) and (TestPendingWrite = True) Then
368	' Write was Successful, return a TRUE
369	TagGroupWPVWithRetries = True
370	
371	Else
372	
373	
374	'Only keep trying if
375	'        1) still have retries left,
376	'        2) the past error was recoverable),
377	'        and 3) haven't had success yet
378	While (RetryCount > 0) And (IsCommErrorRecoverable(Err.Number, , ReferenceString & "WritePendingValues") = True) And
379	(RetrySuccessful = False)
380	
381	Err.Clear
382	
383	
384	
385	'Since this is a retry, take a defined break.
386	'Note that all vba will be paused during this 'Sleep'
387	'so keep the values to a minimum where possible
388	Sleep RetryDelay
389	
390	TestPendingWrite = oGroup.WritePendingValues(MissedTags) ' Retry attempt
391	
392	If (Err.Number = 0) and (TestPendingWrite = True) Then
393	TagGroupWPVWithRetries = True



394	RetrySuccessful = True
395	
396	End If
397	
398	RetryCount = RetryCount - 1
399	
400	Wend
401	
402	'Ok , we're done with the retries.
403	'At this point we would be Successful and will return True
404	'or will return FALSE. If that's the case, at least report it
405	If RetrySuccessful = False Then
406	LogDiagnosticsMessage "Unsuccessful Retry WritePendingValues for " & _
407	"(" & ReferenceString & ")", ftDiagSeverityError
408	
409	End If
410	
411	End If
412	
413	End If
414	
415	Err.Clear
416	
417	End Function
418	
419	-----
420	
421	<b>Public Function IsCommErrorRecoverable(ByVal ErrState As Long, Optional ByRef oTag As Tag = Nothing, Optional ByVal</b>
422	<b>ReferenceString As String = "")</b>
423	
424	On Error GoTo ErrHandler
425	
426	Dim DetailString As String
427	
428	If oTag Is Nothing Then
429	



430	DetailString = " , Last error# 0x" & Hex(ErrState)
431	
432	
433	Else
434	
435	DetailString = "for tag" & oTag.Name & " , Last Tag Error# 0x" & Hex(oTag.LastErrorNumber) & _
436	":" & oTag.LastErrorString
437	
438	End If
439	
440	Select Case ErrState
441	
442	
443	Case 0:
444	'No error occurred
445	'in this context, a success is considered a recoverable state
446	IsCommErrorRecoverable = True
447	
448	Case tagErrorReadValue:
449	'In some cases, this can be as simple as the initial value
450	'has not been retrieved from the device yet (too soon).
451	'You may instead want to consider a short delay and a retry.
452	LogDiagnosticsMessage "Unsuccessful Retry " & ReferenceString & " " & DetailString & " Error# 0x" &
453	Hex(oTag.LastErrorNumber) &
454	":" & oTag.LastErrorString, ftDiagSeverityAudit, ftDiagAudienceEngineer
455	
456	IsCommErrorRecoverable = True
457	
458	Case tagErrorReadOnlyAccess:
459	'If the Client is only Read-Only, no number of retries will help
460	LogDiagnosticsMessage "Unsuccessful Retry " & ReferenceString & " (ReadOnly Client rights) " & _
461	DetailString, ftDiagSeverityWarning, ftDiagAudienceEngineer
462	
463	IsCommErrorRecoverable = False
464	
465	Case tagErrorWriteValue:



466	'There are various reasons a write could have failed. Most are recoverable,
467	'but most of these codes are decided on and generated by the data server.
468	'What is considered unrecoverable (ie. don't waste time on retries) can vary
469	'each with application. In this example code it defaults to allowing additional retries.
470	LogDiagnosticsMessage "Unsuccessful Retry "" & ReferenceString & "" " & _
471	DetailString, ftDiagSeverityAudit, ftDiagAudienceEngineer
472	
473	
474	If oTag Is Nothing Then
475	IsCommErrorRecoverable = True
476	Else
477	If oTag.LastErrorNumber = tagErrorInvalidSecurity Then
478	IsCommErrorRecoverable = False
479	Else
480	IsCommErrorRecoverable = True
481	End If
482	End If
483	
484	Case tagErrorOperationFailed:
485	
486	LogDiagnosticsMessage "Unsuccessful Retry "" & ReferenceString & "" " & _
487	DetailString, ftDiagSeverityWarning, ftDiagAudienceEngineer
488	
489	IsCommErrorRecoverable = True
490	
491	Case tagErrorCollectionIndex:
492	
493	LogDiagnosticsMessage "Unsuccessful Retry "" & ReferenceString & "" " & _
494	DetailString, ftDiagSeverityError
495	
496	IsCommErrorRecoverable = False
497	
498	Case Else
499	
500	'at minimum, report the error
501	LogDiagnosticsMessage "Unsuccessful Retry "" & ReferenceString & "" " & _



502	DetailString, ftDiagSeverityWarning, ftDiagAudienceEngineer
503	
504	IsCommErrorRecoverable = True
505	
506	
507	End Select
508	
509	Exit Function
510	
511	ErrorHandler:
512	
513	'something unexpected happened. Record it and consider this a nonrecoverable situation
514	LogDiagnosticsMessage "Error occurred in IsCommErrorRecoverable(), Error# 0x" & Hex(Err.Number) & ", " & Err.Description, _
515	ftDiagSeverityError
516	
517	IsCommErrorRecoverable = False
518	
519	
520	End Function
521	
522	