



Recommendations For Writing VisualBasic for Applications (VBA) Code In RSVIEWSE

This document is to highlight common errors, mistakes and general advice when writing VBA code for RSVIEWSE. In the document below, when the acronym VBA is referred to, it means explicitly VBA code for RSVIEWSE. Also refer to the extensive Microsoft help on the web for VBA coding help and guidance.

1. Error Checking

Error checking must be enabled for all VBA code routines. The type of error checking that should be used is based on the use of the VBA code in the routine. Having VBA code with no error checking is at some point in time going to cause a problem, (even in the simplest routines), having a problem in the VBA code is going to cause problems with the RSVIEWSE Client or even the overall application.

When designing the error trapping try to think about the following:

- What errors can I test for before an error occurs, e.g. when doing arithmetic could you get a division by zero, so test for zero first.
- What errors may occur but I should expect them? E.g. writing to a drive that maybe is a floppy drive and is full.
- What to do if an error occurs that I did not expect?
- If an error does occur can operator intervention fix it?
- How to prevent fatal errors that will crash the VBA code.

In VBA you have the options of using:

'On Error Resume Next'

'On Error Goto *Label*'

Do not use the 'Resume' when returning from an error trap, this is in fact worse than not error checking at all, since it can cause an infinite loop to occur, if the line in execution has an error, causing 100% CPU and crashing.



The 'On Error Resume Next' should only be used in limited circumstances, the reason for this is that if an error has occurred on a line of VBA code, you are continuing in an unknown state. However in limited cases where you do not care if an error occurred or not then this can be used, usually only for very short VBA code routines.

The recommended way is to use the 'On Error Goto *Label*'. It is then possible to control either at a routine level or higher level what the errors is, if you expected it and what to do about it.

As an example: The idea of the ErrorPosition variable is that you can make a reference as to what VBA code is about to be executed, so that errors can be categorized and the handling of them changed accordingly.

On Error GoTo MyErrorTrap

Dim ErrorPosition as Long 'Use this variable as an index to the position in VBA code.

ErrorPosition = 1

 Some VBA code Lines

ErrorPosition = 2

 Some More VBA code Lines

ErrorPosition = 3

 Etc.

Exit Function or Sub

MyErrorTrap:

'Log error to FT Event File for later analysis. To aid fault finding include the DisplayName.GFX, the Function/Sub routine name and the ErrorPossition code to isolate where the error occurred.

LogDiagnosticsMessage "Graphic: <filename>, Function: <function name>, Code Position: " & _
ErrorPosition & " Error Code : [" & Hex(Err.Number) & "], Description: " & Err.Description

Select Case ErrorPosition

 Case 1

 Decide What To Do, Were to return in VBA code.

 Case 2



Decide What To Do, , Were to return in VBA code.

Etc

End Select

End Function or Sub

IMPORTANT: The VBA code in the error trap routine must **not** have an error itself or the VBA code will crash, so keep it simple

2. Keep it Short, Keep It Simple, Keep it Fast And Don't Keep Calling It.

VBA code is not a separate VisualBasic (VB) '.exe' application. VBA runs in conjunction with an application in this case RSVIEWSE. The VBA code routines are there to enhance the RSVIEWSE application, not to write the whole application. Keeping the routines short and simple helps with understanding of the SE Client VBA code and speed of execution. Also keep the VBA code fast, do not use long loops or the DoEvents events instruction (see: Point 11 Below), if small delays are required then use the Sleep API call (keep any delay short, the Sleep API blocks the VBA thread). It is also important not to call routines at very high rates of execution, even if short and simple.

VBA code should be like a commando: Only use them when you really have to; Get in, get the job done fast and get back out again.

3. If You Can Do It In RSVIEWSE, Do It There, Not In VBA

If the functionality of what you are trying to achieve can be achieved within RSVIEWSE standard functionality, then, that is the place to achieve it, do not use VBA code. This includes such things as animation, colors, user input, event files, derived tags etc.

4. Option Explicit – Is Not An option

The use of the 'Option Explicit' is not optional; use it to catch any typo problems with your variables before they cause trouble in the VBA code.

5. No Comment

'No Comment' might be an accepted statement from a politician, but not for VBA code writers. Add meaningful and helpful comments to your VBA code to help others and yourself understand; what, why and how you are doing, what you are doing. Also put some revision control in there, no one is going to think any less of you because you are up to V15.4



6. Tidy Up After Yourself

When setting references to objects ensure they are set to 'nothing' when you are finished with them. Also ensure that objects are set to 'nothing' in the reverse order that you made reference to them, if the object is dependant on the one before.

Dim gpGroup as TagGroup

Dim tgTag as Tag

Set gpGroup = Application.CreateTagGroup(Me.Areaname)

Set tgTag = gpGroup.Item("TagName")

'VBA code here

Set tgTag = nothing

Set gpGroup = nothing

7. Live And Let Die

If you create a reference to an object (as in 6 above for a tag group and tag), and you intend reusing the reference repeatedly whilst the SE Client is running, then don't kill the object until the SE Client is stopped.

For example:

- If your objects are required for the duration of the display being open, then declare them globally and de-reference when the display is closed.
- If your objects are required for the duration of the SE Client life, then declare publicly on a cached hidden display (so the display is not closed until the SE Client is stopped) and de-reference only when the display closes as the SE Client is requested to stop.



8. In One FORM Or Another.

If possible avoid using VBA Forms. The idea of VBA is to enhance the functionality of RSViewSE (which is after all visualization software), so use RSView SE displays and not VBA forms. Some reasons for not using forms include: depending on the type of form it can block the VBA calls and events, you have another visual interface to handle and ensure the operator clicks on the form and not back on the RSViewSE display, the SE Client VBA code is harder to maintain if there are more places apart from RSViewSE displays that can be have VBA code to be stored and called.

If you really have to use a VBA form, use Modeless forms, and don't use ActiveX controls on the form (apart from the VBA controls), as you may well not get the functionality you expect. Put the ActiveX's on the RSViewSE display.

9. Got The Message

Don't call the Message Box (msgbox) to display messages or errors, they block your VBA calls and get in the way, use the:

LogDiagnosticsMessage "Message " & 'Message To Display'

or call a RSViewSE display with your message in a string tag if you want it to be really obvious).

10. To Database Or Not To Database.

VBA code is frequently used to interface to databases for retrieving and storing data. This is fine but recommendations still apply when doing this type of work. There are three main things to be aware of here, assuming that you are already familiar with VBA and database interfacing:

- Frequently the database is located on a separate PC to the one where your VBA code is running, this means a network is involved and hence potential disconnections and other network related issues, these can cause hang-ups and delays in the VBA code when accessing database objects.
- Whilst developing SE Client VBA code, the access to the database objects may be very fast and not cause you any concern; however after the SE Client has been running for a while; databases if not managed or index correctly can grow to a size where significant delays are experienced, and hence response and hang-up problems can occur.
- Be ready to handle database table or record locks. These can occur in several situations, particularly multi-user environments. It is necessary to decide on the best action; retry, error etc.



11. Sleep Perchance To Dream

Using long loops in the VBA code can produce unresponsive and problematic VBA code. One temptation here is to use the DoEvents keyword to return control to the system temporarily. If at all possible try to find an alternative solution to the use of DoEvents.

The reason for not using 'DoEvents' are as follows: It can cause unpredictable, unresponsive and recursive behavior (out of stack space) problems. Two very bad scenarios of using DoEvents are; 1. It is possible for the operator to close the SE Client display and hence stop the VBA code executing immediately at the current execution point, hence not exiting the VBA code in a known state. 2. Re-entry in the VBA code can give some unpredictable results if global objects are used e.g. tag values. If DoEvents must be used then additional coding is required to prevent the above problems.

If you need a time delay use the Sleep API call, but keep it as short as possible and fully test the VBA code before production use. The Sleep API call will block code execution, but will not stop queued events. If delays are too long the operator can click many times on objects on the SE Client that may cause issues that you were not expecting.

12. Keep Up On Current Events

The use of the With Events keyword gives some very useful and powerful potential to your application, just be sure that your application can handle the maximum rate that events can occur; otherwise you might find the event VBA code is being called at a frequency that can give you problems.

Pay particular attention when using - Tag Value Change Events. Here VBA code execution is controlled by tag values changing not operator interaction, so high tag rate values changes will trigger VBA code repeatedly.



13. I've Told You Once.

Always verify that an object does not exist before creating.

For example:

Dim obj as Object

If obj is nothing = False then set obj = nothing

Now create your object.

This will assure that the object is in a known state. This is useful when you want to change the behavior of an object and then recreate.

14: Speed Kills

Use the slowest time updates rates that are acceptable, this is true for all RSView SE objects (Alarms, Derived Tags, Events etc) including VBA objects, such as tag groups. Look at the whole application and only update items at a rate that is sensible for the type of application.

15. To Play Tag You Have To be Fast.

Accessing HMI or Direct Reference Tag values and properties in VBA code takes considerably more time, relative to accessing say a local variable. So if possible use local variables wherever you can. If looping try to use local variables and then assign to a Tag Value outside of the loop.

16. If You Build It, They Will Come.

When building your VBA code, think about re-use. Try to make any VBA code that is useful, reusable by yourself and others.

ERROR: syntaxerror
OFFENDING COMMAND: --nostringval--

STACK:

/Title
(
/Subject
(D:20080505174115)
/ModDate
(
/Keywords
(PDFCreator Version 0.8.0)
/Creator
(D:20080505174115)
/CreationDate
(acrossla)
/Author
-mark-