

RATIONAL ROBOT
SCRIPTING
STANDARDS

1. Naming Convention

1.1 Local scope variables

Description

Variables represent values that can be changed within a procedure or function. Local scope variables are placeholders that reside within a function- or a script-body.

Syntax

Use a lower case prefix for the type.

Concatenate scope and type prefixes, with the following precedence:

[Prefix]+[ShortDescription]

[Prefix] is a lowercase letter (either "n", "s", or "d" appropriate to the type it represents)

[ShortDescription] is a corresponding name of what the variable stands for.

If [*ShortDescription*] consists of more than one word they are all separated using a capital letter for each new word.

Type	Prefix	Example
Double	d	dQualityFactor
Integer	i	iRowCount
Long	l	lLongDistance
String	s	sFileName
Variant	v	vPropertyValue
Object	o	oExcelWorksheet
Array	a	aObjectProperties()
Global	g	GStatus
Boolean	b	bChecked
String array	sa	saObjectData()
Global variant	gv	gvItemIndex

*Variable names should be descriptive and not ambiguous. E.g.

x	fname
✓	sFileName

1.2 Module-level variables

Description

If variables are placed outside a function body their scope will be different from a local scope variable, therefore we flag those variables with a prefix "m_".

Syntax

"m_" + [Prefix]+[ShortDescription]

Examples

- m_nCountDatabaseRecords
- m_strLastname
- m_sPassword

1.3 Arrays

Description

Within SQABasic the size for an array can either be fixed or grow if the number of items in the list is not known during compilation of the source code.

Syntax

"a" + [Prefix]+[ShortDescription]

Letter "a" indicates that the variable is of type array.

[Prefix] is a lowercase letter that represents the type of variables in the array.

The rules for *[Prefix]* are the same as for "Local scope variables".

Examples

- asPersonList
- anNumberList

1.4 Globals

Description

The values of global variables can be used and changed all over the project within all scripts and libraries. Though it is highly recommended to keep the number of global variables small. If global variables tend to be used in a specific project only it is advisable to keep those values in either a project specific library header or in a separate configuration file.

Syntax

"g" + [Prefix]+[ShortDescription]

Letter "g" indicates that the scope of the variable is global.
[Prefix] is a lowercase letter that represents the type of the global variable.
The rules for *[Prefix]* are the same as for "Local scope variables".

Examples

- gnNumOfPersons
- gsPersonLastname

1.5 Constants

Constants are “variables” that cannot be changed within a function- or script-body. The value will always be the same during script-execution.

Constant names should be descriptive and not ambiguous. Constant names should be in upper case, with proper words separated by the underscore character.

Syntax

"AX_" + [PURPOSE] + {OPTIONAL}

If the constant name consists of more than one word those will be separated using an underscore.

Examples

x	Const FatalRateOfDescent = 1
✓	Const AX_FATAL_RATE_OF_DESCENT = 1

Constants should be used to improve the readability and maintainability of the source code.

Examples

x	TabControl Click, "Name=tabMain;ItemText=Album"
✓	Const ALBUM_TAB = "Name=tabMain;ItemText=Album" TabControl Click, ALBUM_TAB

2. Verification points

Description

In functional testing, verification points are required to verify that the objects in the application-under-test look and work as designed from build to build.

To accomplish this, you can establish verification points also known as checkpoints or the objects.

Because verification points are not very flexible when it comes to changes in the AUT it is wise to prefer Robot's **SQAGetProperty** - commands to verify whether expected behavior is met or not.

Result=SQAGetProperty("Name=frmLogin","FullRecognition",value)

'This syntax above will verify that the Login Form has loaded. This is verify important if we want to only verify that the login form has loaded.

Result=SQAGetProperty("Name=frmLogin;;Name=txtPassword","FullRecognition",value)

'This syntax above will verify that the Login Form has loaded and we can see the Password text box.

Syntax

[vp]+[FEATURE]+ [FUNCTION]

Examples

- vpAURA_ImageUploaded
- vpAURA_CaseInserted

3. Functions

Description

Functions abstract a sequence of SQABasic code statements in either a script-file (.REC) or a library-file (.SBL) and cover them with a reasonable name that is explanatory to others.

Syntax

[SOURCE]+"_"+"[PerformedAction]+[Parameterlist]+" As " + *[ReturnValue]*

[SOURCE] is an abbr. of the library name it is implemented at.

[PerformedAction] consists of two words, an action(verb) and an object the action is performed against.

[Parameterlist] is a sequence of placeholders the function is feeded with.

[ReturnValue] is the value returned by the function

Examples

- □UTIL_CountItemInString(sHaystack\$, sNeedle\$) As Integer
- SERVICE_Start(sServiceName\$) As Integer
- □SERVICE_StopAllServices() As Integer

If the list of parameters within a function declaration is too long, insert line breaks using underscores

```
// BEVOR in den Content Manager eingeloggt wird, wird überprüft ob ein Exp
'// Dabei werden die Funktionen GUI_IsReadyWindowExistence, GUI_IsReadyWind
'// *****
'//
Function CL_LoginContentManager(strContentManagerUrl As String, _
                               strWindowName As String, _
                               strUserID As String, _
                               strPassword As String, _
                               nErrorMessage As Integer) As Integer
|
on Error goto Err_CL_LoginContentManager

    Dim nReturn As Integer : nReturn = False
```

4. Formatting

Indentation

Standard (tab-based) block nesting indentation should be used.

Code in all procedures should be indented one tab stop.

Code within programming constructs such as

If...End If, Select Case...End Select, For...Next, Do...Loop and While...Wend should be indented one additional tab stop.

Nested constructs should be indented an additional tab stop for each level of nesting.

Example:

```
For iRow = 1 to iRowCount
    For iCol = 1 to iColCount
        Select Case saArray(iRow, iCol)
            Case "Alpha"
                iAlphaCount = iAlphaCount + 1
            Case "Beta"
                iBetaCount = iBetaCount + 1
        End Select
    Next
Next
```

5. Comments

All functions, procedures or like elements of a source file should have a header block which provides a brief description of the functional characteristics of the routine.

The code itself and any necessary in-line or local comments should describe the implementation of the functional characteristics. As a rule of thumb, reading the comment lines alone should adequately describe the program flow.

Source code lines should have local or in-line comments following these guidelines:

- Identifying each independent function within the code
- Unusually complex statements such as deeply nested statements
- Statements which are temporary in nature (such as for testing or debugging)
- Where values are "hard-coded" other lines which may not be obvious to any likely reader

6. Flow Control

IF

1. Put the normal case after the *If* rather than after the *Else* clause. Normal processing should occur immediately after the *Then* clause.
2. Follow the *If* clause with a meaningful statement. Do not code null *Then* clauses to avoid using *Not*. Example:

Incorrect	Correct
<pre>If EOF(iFileNum) Then ' do nothing Else ProcessRecord End If</pre>	<pre>If Not EOF(iFileNum) Then ProcessRecord End If</pre>

3. An *Else* clause is usually recommended.
4. Simplify complicated conditions with Boolean function calls. Rather than test multiple conditions in a single *If* statement, create functions that return True or False for specific conditions.
5. Do not use chains of *If* statements if a *Select Case* statement can be used.

7. SELECT CASE

1. Put the normal case first.
2. Order cases by frequency of occurrence.
3. Keep the actions of each case simple. If necessary, create functions or procedures that are called from each case.
4. Use *Case Else* only for legitimate defaults, not to avoid coding a specific test.

DO

1. If possible, keep the entire body of a loop visible on the screen at once.
2. Limit nesting to three levels.

8. FOR

Do not use i, j, k et al as loop counter variables. Use descriptive names, as documented under Variable Naming Conventions. Example:

x	For i = 1 to iRowCount For j = 1 to iColCount
✓	For iRowIndex = 1 to iRowCount For iColIndex = 1 to iColCount

Do not omit the loop variable from the Next statement. It is easier to debug loops if their end points identify themselves properly. Example:

x	For i = 1 to iRowCount For j = 1 to iColCount Next Next
✓	For iRowIndex = 1 to iRowCount For iColIndex = 1 to iColCount Next iColIndex Next iRowIndex

9. Syntax Checks

The following checklist summarizes the conventions documented above. In addition, it references the function that tests compliance with the convention in the CodeCheck utility library.

Convention	Test Function
1. Option Directives	
1.1 Option Explicit should be used.	CheckOptions
1.2 The Array Base (lowest boundary) should specified and documented in the code. Where this is 0 or 1, the Option Base directive should be used.	CheckOptions
2. Declarations - Functions & Procedures	
2.1 All functions and procedures within a script or library, with the exception of Sub Main, should be declared before the first function or procedure.	CheckDeclare
3. Declarations - Variables	
3.1 All local variables should be declared at the beginning of a function or procedure.	
3.2 Variable declarations should be explicit.	CheckTypeSpec
3.3 Variable names should be descriptive and unambiguous.	
3.4 Variable name prefixes should indicate the datatype	CheckTypePrefix
3.5 Variable names should use "camel" notation.	
3.6 Unused variables should not be declared.	
3.7 Type-declaration characters (e.g. \$ for string, % for integer) should not be used.	CheckTypeChar
3.8 Meaningful loop index names should be used.	
3.9	
4. Declarations - Data Types (refer to Declarations - Variables)	
4.1 Data type names should be descriptive and unambiguous.	
4.2 Data type name prefixes should indicate the datatype.	

Convention		Test Function
4.3 Data type names should use "camel" notation.		
4.4 Unused data types should not be declared.		
4.5		
5. Constants		
5.1 Constant declarations should be explicit.		
5.2 Constant names should be in upper case.		
5.3 Proper words in constant names should be separated by underscores.		
5.4		
6. Formatting		
6.1 Standard (tab-based) block nesting indentation should be used.		
6.2 White space (blank lines) should be used to indicate logical program sections.		
6.3 There should be no more than one statement per line.		
6.4 Long statements should be broken into multiple lines using the line-continuation character.		
6.5 All functions and procedures should have a header block describing the functional characteristics.		
6.6 In-line comments should describe logical program sections or complex statements.		
6.7		
7. Expressions		
7.1 Use of "&" in place of "+" for string concatenation.		
7.2		
8. Initialization		
8.1 Each routine should check input parameters for validity, wherever appropriate.		
8.2 Counters and accumulators should be initialized before use.		
8.3 Variables should be initialized before re-use.		
8.4 Variables should be initialized close to the statements in which used.		

Convention		Test Function
8.5		
9. Comparisons		
9.1 Comparisons to named constants should ensure the variable or expression will result in the same data type as the named constant.		
9.2 When comparing strings, the leading and trailing blank characters should be trimmed and the string converted to upper case.		
9.3		
10. Flow Control		
10.1 It is <i>recommended</i> that each function or procedure has an error handling routine.		
10.2 The <code>GoTo</code> statement should not be used, except within an <code>On Error</code> clause.		
10.3 The <code>GoSub</code> statement should not be used.		
10.4 The <code>stop</code> statement should not be used.		
10.5 Independent groups of statements and those that are re-used should be packaged into their own routines.		
10.6 Single-line <code>If ... Then</code> statements should not be used.		
10.7 In an <code>If ... Else ... End If</code> statement, the normal case should follow the <code>If</code> rather than the <code>Else</code> clause.		
10.8 Each loop should perform only one function.		
10.9 Nesting should be minimized [restricted to n levels?].		
10.10		
11. File Access		
11.1 The mode should be specified in an <code>open</code> statement.		
11.2 A literal number should not be used for the file number in the <code>open</code> statement. A variable should be declared and assigned using the <code>FreeFile</code> function.		
11.3 All files opened should be explicitly closed with the <code>close</code> statement.		
11.4 Path names should not be "hard coded". Where appropriate, provision must be made to enable the		

Convention		Test Function
routine to ascertain or permit the user to select any required file path information.		
11.5 Mapped drive letters should be avoided unless provided as input from the user.		
11.6 All path names should use the Universal Naming Convention (UNC) (\\server\share\directory) for shared drives.		
11.7		
12. General Syntax		
12.1		
12.2		
13. Functions and Procedures		
13.1 Functions and procedures (except Sub Main) should be declared before the first function or procedure.		
13.2 Functions should have a type specification and return a value on every exit path within the function.		
13.3		
14. Event Logging		
14.1 Key execution information should be recorded to an event log.		
14.2		
15. Error Handling		
15.1 Each script / module should have an error handling routine for unforeseen errors.		
15.2		
15.3		