

Test Script Standards

Coding standards for Rational Robot

| | |
|--------------|------------------------|
| AUTHOR | Torsten Zelger |
| VERSION | V 1.0 |
| VERSION DATE | 22.02.2004 |
| FILE NAME | CodingStandardsAUT.doc |

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction: | 1 |
| 2 | Naming Convention | 1 |
| 2.1 | Local scope variables | 1 |
| 2.2 | Module-level variables ("member-vars")..... | 2 |
| 2.3 | Arrays..... | 2 |
| 2.4 | Globals..... | 3 |
| 2.5 | Constants..... | 3 |
| 2.6 | Functions | 3 |
| 2.7 | Scripts | 4 |
| 2.8 | Libraries | 4 |
| 2.9 | Verification points..... | 5 |
| 3 | Functions | 6 |
| 3.1 | Introduction | 6 |
| 3.2 | Parameters | 6 |
| 3.3 | Indentation | 6 |
| 3.4 | Return value..... | 6 |
| 3.5 | Error handling | 8 |
| 3.6 | Size | 8 |
| 4 | Scripts and libraries | 9 |
| 4.1 | Introduction | 9 |
| 4.2 | How to deal both scripts and libs | 9 |
| 4.3 | Size | 9 |
| 5 | Documentation | 10 |
| 5.1 | Introduction | 10 |
| 5.2 | Functions | 10 |
| 5.3 | Scripts | 10 |
| 5.3.1 | Header | 11 |
| 5.3.2 | Public dependencies..... | 11 |
| 5.3.3 | Forward declaration | 11 |
| 5.3.4 | Module level variables | 12 |
| 5.3.5 | Body | 12 |
| 5.4 | Libraries | 12 |

1 Introduction:

The standards are for all testers using the IDE of Rational Robot to develop their automated test scripts.

The mission is to reduce maintenance costs when it comes to changes. This is done by setting up rules on how to name variables, functions, scripts and libraries, where to place them and when to use them.

The standards do not provide any information about the use of frameworks, best practices, Rational Robot settings and version control. These are all provided in separated documents.

2 Naming Convention

2.1 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

[Prefix]+[ShortDescription]

[Prefix] is a lowercase letter (either "n", "s", "str", "d" or "t" 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.

| Prefix | Datatype | Examples |
|----------|---|---|
| n | integer, short, long, float | nCountNumberOfRecords nNumItems nValue |
| s or str | string | sLastname sUsername sPassword strObjectRecognition |
| d | Date | dToday dTomorrow dSomeday |
| t | abstract datatype also known as structs | tPerson |
| v | Variant | vValueFromDB |
| o | Object | oExcel, oFile |
| a | arrays | AsPersonList |

2.2 Module-level variables ("member-vars")

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_" that is very common in object oriented languages even though SQABasic is not an object oriented programming language.

Syntax

"m_" + [*Prefix*]+[*ShortDescription*]

[*Prefix*] is a lowercase letter (either "n", "s", "str", "d" or "t" 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 these are all separated using a capital letter for each new word.

Examples

- m_nCountDatabaseRecords
- m_strLastname
- m_sPassword

2.3 Arrays

Description

Arrays in many programming-languages usually represent a fixed list of values (e.g. a list of lastnames). However 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

2.4 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

2.5 Constants

Description

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

Syntax

"AX_" + [PURPOSE] + {OPTIONAL}

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

Examples

- AX_AUT_VERSION
- AX_DEFAULT_URL

2.6 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

The function name must always correspond to what the function is actually doing. That is the script developer knows what the function is doing without the need to dig any deeper into its implementation. If the function name is called "NaviagteTo("a-web-link") it is expected that the function does only navigate to that specific web-link and not doing additional actions that is deletion and/or creation of any records in the DBMS.

2.7 Scripts

Description

A script within Rational Robot is a file that contains a sequence of SQABasic code. The extension of the file is always ".REC".

Syntax

[FEATURE] + "_" + [FUNCTION] + "_" + {optional}

[FEATURE] is a capital letter string denoting the feature name

[FUNCTION] is a capital letter string denoting the function

{optional} means that the remaining characters preceded by an underscore will be optional and left to each person to define a sensible name which clearly identifies each of the test scripts

Example

- BILL_REP_Outputformat (refers to script Billing feature with report function)
- PART_INS_Assesor
- PART_UPD_Assessor

2.8 Libraries

Description

Common used functions are placed in libraries. These are located in the SQABas32 subdirectory of the Robot working directory. A library is divided into three files, a header (.SBH), an implementation file (.SBL) and the compiled version (.SBX). Libraries are not necessarily bound to an AUT or feature.

Syntax for implementation file

[ax]+[ShortName]+".sbl"

Syntax for header file

[ax]+[ShortName]+".sbh"

Examples

- axCommonUtilities.sbl
- axDBAccess.sbh
- axGuiMapper.sbl

2.9 Verification points

Description

In functional testing, you need 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.

Syntax

[vp]+[FEATURE]+ [FUNCTION]

Examples

- vpAURA_ImageUploaded
- vpAURA_CaseInserted

3 Functions

3.1 Introduction

Common used functions are placed within libraries in the SQABasic directory. Files ending with ".SBH" contain the public interface they provide to other libraries and scripts. Files ending with "SBL" contain the implementation of the public interface and private functions.

You use them by including them in your testscript or in another library. However, never ever include a .SBL-file into your script or library, instead include the header file only (.SBH) otherwise you run into cyclic redundancy problems and the compiled SBX file grows for nothing.

Example

GOOD:

```
'$include "axCommon.sbh"
```

FORBIDDEN:

```
'$include "axCommon.sbl"
```

3.2 Parameters

If the list of parameters is too long, insert line breaks using underscores (see example below)

```

// BEVOR IN DEN CONTENT MANAGER EINGELOGGT WIRD, WIRD UBERPRÜFT 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

```

Although it may have an impact on readability within other editors use tabs instead of blanks for indentation because the latter slows down development of scripts. By the way notepad shows the indentation of robot scripts correctly.

3.3 Indention

Use tabs to bring some structure into your function body

```

if (nPos > 1) then
    nReturn = True
else
    nReturn = False
end if

```

3.4 Return value

The variable returned at the very bottom of the function body must always be declared and initialized in the first line of code.

```
Function CL_CheckVersionFrontend() as integer
on Error goto Err_CL_CheckVersionFrontend

    Dim nReturn%:          nReturn = False
    Dim strText$:         strText = GUI_GetPropertyValue("LoginVersionText", "",
    Dim nPosVersion%:     nPosVersion = instr(strText, "Version " & IX_VERSION_MF
    Dim nPosBuild%:       nPosBuild = instr(strText, "(Build " & IX_VERSION_BUIL

    if (nPosVersion > 1) AND (nPosBuild > 1) then
        nReturn = True 'nur dann wird die initialisierte nReturn-Variablen auf 0
    end if

Exit_CL_CheckVersionFrontend:
    CL_CheckVersionFrontend = nReturn
    Exit Function
Err_CL_CheckVersionFrontend:
    nReturn = LOG_Message(-9999, "ERR: Fatal exception in CL_CheckVersionFrontend
    nReturn = False
    Resume Exit_CL_CheckVersionFrontend
_ _ _ _ _
```

3.5 Error handling

All functions need to provide an error catching cover such as

"On Error goto Err_" + [FunctionName]

and an appropriate label to deal with the error.

Although the "goto" statement should be abandoned from any programming languages, this is the one and only place where it is allowed to use it. In order to make it easier for all to follow the below shown function body, a tool called "BodyWizard" is provided that does the skeleton for you automatically.

```

'// *****
'// Logout Cloudbreaker application
'// *****
'//
Function CL_Exit() As Integer
on Error goto Err_CL_Exit

    Dim nReturn As Integer: nReturn = -1

    call GUI_Click("TLBExit", "")
    call GUI_Click("MsgOK", "")

Exit_CL_Exit:
    CL_Exit = nReturn
Exit Function

Err_CL_Exit:
    call LOG_Message(sqaFail, "Fatal exception in CL_Exit()", "an exception occurred")
    nReturn = -1
    Resume Exit_CL_Exit
End Function

```

3.6 Size

A function body should fit within an A4-page (approx. two monitor-pages). If the code does not fit it is a candidate for more decoupling to smaller functions. The larger a function body the harder to read and maintain.

4 Scripts and libraries

4.1 Introduction

Scripts

A script within Rational Robot is a file that contains a sequence of SQABasic code. The extension of the file is always ".REC". Typically the script contains an automated testcase and contains the business-logic and also the testdata. Although it is generally recommended to shift testcase-logic and especially testdata to external sources such as CSV or Excel it is NOT designed to do so today because this technique requires an underlying sophisticated automated test framework that we do not have implemented at this level of automation yet.

Libraries

Common used functions are placed in libraries. These are located in the SQABas32 subdirectory of the Robot working directory. A library is divided into three files, a header (.SBH), an implementation file (.SBL) and the compiled version (.SBX). Libraries are not necessarily bound to an AUT or feature.

4.2 How to deal both scripts and libs

Although it is possible to chain scripts by using the "CallScript" command it is not recommended to use it (or even exhaust) too much unless successful script execution of the called script does not depend on the prior executed script or if the chain is not too long and if a script is preceded by an initiating script followed by a script that brings the AUT back into a defined state.

Functionality or a business-case that is used very often such as the login-operation should not be placed in a script but in a function library because scripts don't allow for parameter passing, ie. a login-script is unable to be reused with different usernames and passwords.

4.3 Size

Main script section ("Sub Main .. End Sub) and function bodies should fit within an A4-page (approx. two monitor-pages). If the code doesn't fit it is a candidate to do more decoupling to separate small functions. The bigger a function body the harder to read and maintain.

5 Documentation

5.1 Introduction

Documentation is done to provide others with information and ease maintenance. The best documentation is done in the headers (function and scripts) and directly in the code. Any useful thoughts ie a chosen algorithm or a solution to a specific problem should be documented in order to help others understand the script(s) and/or function(s).

5.2 Functions

All functions within libraries and/or scripts need explanatory headers where one describes the purpose of the function.

```
'// *****  
'// Logout Cloudbreaker application█  
'// *****  
'//  
Function CL_Exit() As Integer  
on Error goto Err_CL_Exit
```

5.3 Scripts

Scripts (.REC-files) always have a header followed by a list of dependencies/includes, forward declarations and module-level variable declarations before the main section of the script starts.

5.3.1 Header

```

'//-----
'//
'// Script:      PART_INS_Many.rec
'//
'// AUT:        APG2 Versions 1.x, 2.x, 3.1
'//
'// Purpose:    The purpose of this script is to test the insertion of a partner
'//
'// Precond:    [ Script expects that a user with administrative rights is already
'//             logged in. Menu=Mainmenu
'//
'// Postcond:   See Precond.
'//
'// Notes:     Simple testcase [ no data oriented testcase ]
'//
'//
'// History:    Feb-20, 2004: Torsten Zelger
'//             created
'//             Feb-21, 2004: Rupa Bannur
'//             fixed bug 3442
'//             Feb-22, 2004: Daniel Reimann
'//             reviewed and renamed variable meet guidelines
'//
'//-----
'//

```

| | |
|-----------|--|
| Script: | Name of script |
| AUT: | Application-under-test and version(s) it is running against |
| Purpose: | Summary of what is the script doing |
| Precond: | Preconditions that must met before script is running |
| Postcond: | What's the final state of the application after the script did or did not execute successfully |
| Notes: | Additional information |
| History: | Date and name of authors and changes |

5.3.2 Public dependencies

```

'// public dependencies
'//-----
'$include "global.sbh"
'$include "ixGuiMap.sbh"
'$include "ixBrowser.sbh"
'$include "ixLogging.sbh"
'$include "axCommon.sbh"

```

5.3.3 Forward declaration

```

'// forward declaration
'//-----
'//
Declare Function DoSomething($WithIt$) As Integer
Declare Function DoNothing($WithIt$) As Integer

```

5.3.4 Module level variables

```

'// module-level variables ("member"-vars)
'// -----
'//
Dim m_nCountItems%
Dim m_sLastname$

```

5.3.5 Body

```

'// start
'// -----
Sub Main

    'Initially Recorded: 13.02.04 16:33:44
    'Script Name: TestTrash

    call BROWSER_StartBrowser("http://www.audatex.ch")

    call GUI_Click("LinkLanguages", "")
    call GUI_Click("LanguageGerman", "")
    call GUI_Click("SearchButton", "")
    call GUI_EnterData("SearchField", "", "audapad")
    call GUI_Click("SubmitButton", "")
    call GUI_Click("SearchResultAudapad", "")

    Dim sValue$: sValue = GUI_GetPropertyValue("SearchResultList", "", "innerText")

    if (instr("15 records found", sValue) > 0) then
        LOG_Message(sqaSuccess, "OK")
    else
        LOG_Message(sqaFail, "Error", "This script failed because...")
    end if

End Sub

```

5.4 Libraries

The documentation rules are almost exactly the same as for Scripts "Documenting scripts". Although from a technical point of view it is feasible to place procedures in libraries it is not common to do that. Functions are a better alternative as they accept parameter passing.