PROGRAMMING STANDARDS

Master List Ref.: PR-ISMS-TKM-014	Issue Date: 5-Jan-2010	
Version No.: 2.0	Issued by: Software Team	Approved by: DGM (Software)

CONTENTS

1 Introduction

The purpose of this document is to provide a basic guideline for developing VB applications, modules, solutions and classes in Rev IT. This document is a working document and will be added to over time.

2 Naming Conventions

One useful tool in making code readable and maintainable is the use of a naming convention. When implemented properly, it allows a developer to quickly follow any block of code. In programming under VB.NET, there are a number of objects that are commonly used. Each of these items will be discussed below.

2.1 Form Elements

Form elements are objects that used to interact with the user via the Graphic User Interface. The table below describes the naming conventions that we will use when building forms.

Naming Convention for Form Elements

Form Elements	Prefix	Example
Label Field	lbl	lblName
Text Field	txt	txtName
Listbox	lb	lbMsg
ComboBox	cmbo	cmboDocType
Menu	mnu	mnuExit
Push Button	btn	btnOK
Form	frm	frmMainGUI
Check box	cb	cbName
Radio Button	rb	rbName
Data Grid	dg	dgGrid
Checked List box	clb	clbBatches
List View	lv	lvFiles
Tree View	tv	tvFiles
Tab Control	tb	tbPanel1
Progress Bar	pb	pbComplete

NOTE: The name that follows the prefix should relate to what is stored in the item.

2.2 Solution File(s)

A solution can be made up of one or more source file(s). The table below describes what naming convention is to be used for the source files.

Naming Convention for Solution files

Form Elements	Prefix	Example
Form Source	Frm	frmBALoad.vb
Module Source	Mod	modBAgen.vb

2.3 Database Columns

If the development project is making use of any database, the column names will have the following format.

Naming Convention for Database Columns

Form Elements	Prefix	Example
String	s	sName
int16	i	iDocCnt
int32 or int64	l	lSysKey
Date/Time Field	dt	dtStartDate

2.4 Properties of a Class and/or local variables

The properties of a class have the following naming convention. NOTE: The naming conventions of properties and/or local variables will not be as string. It would be good to follow these conventions but is not mandatory.

Naming Convention for Class Properties and/or Local variables

Form Elements	Prefix	Example
Public	g	gName
Private	My (or not g)	myName
Boolean	b	bDocumentOpened
String	s or str	sName strName
int16	i	iCnt
int32 or int64	l	lCnt or glCnt
Date	Dt	dtCurrent or gdtCurrent

3 Methods

Methods are routines/actions that are associated to a class. A method will “operate” or “manipulate” properties. There are two types of methods, subroutines and functions. Sub routines do not pass/return any values, functions do so.

3.1 FUNCTIONS

Functions are routines that allow the caller and function to pass data via a return code.

3.1.1 ERROR HANDLING

Each function will make use of the Try/Catch method of error trapping. An example of the standard function is shown below:

```
Public Function AnExample( byval strName as String ) as Int32
 Dim rc as int32

   rc = 0

   Try

         Block of good code …..

   Catch
         Rc = 1
         sErrMsg = “AnExample: Unexpected Error(“ & Err.Description & “)”

   End Try

   AnExample = rc

End function
```

3.1.2 RETURNING

Most of the functions used by Rev IT will make use of a return code. The function will be created to return an int32. The return code will return 0 if no error and a non-zero if error. In addition, each class will have as it member a string called sErrMsg. This string will be used to hold the last error code. The detail reason of why a function has failed will be written to this string.

3.1.3 BUILDING OF THE ERROR STRING (CATCH BLOCK)

The catch block is the catch portion of the try/catch block that was described above. The logic only flows to this block when there is a system exception. This is usually caused when an object is not properly used. For example, if you are using xml.net and have tried to load a xml document that is not value, the logic will flow to this block..

The catch block by definition is an exception that should never be reached. If the logic does flow to this block, we need to ensure that an operator/engineer knows about it. In this block, we need to show two things: the function where the event occurred and the description of why it occurred. The error message should be formatted as follows:

FUNCTNAME: Unexpected error(“ & Err.Description & “)”

FUNCTNAME is the name of the function that you are in
Err.Description is a property of the err object. This will explain why you had an exception.

3.1.4 Building of the Error String

There are other exceptions that can be reported that are not extreme violations and would not be caught in the “catch” block. There are two basic types, functions within an application or from external classes that are used by and referred via a DLL.

3.1.4.a Internal to the application

A given application may have many internal functions. These function can them selves call many functions. If an error is encountered within this code (an application error, not one that is generated via the catch block), do the following:

- Write an error message to the sErrMsg property that each application will have.
- Set the return code to a non-zero value

The error message should make sense to the user. Give as much information as possible that will help a user solves a problem. For example, if you are trying to copy a file and the file cannot be copied, include the file that you are trying to copy in the error message.

3.1.4.b External to the application

If you are getting an error returned from an API/Class that is stored in a DLL and this DLL was developed by RevIT, you would need to capture the string name that is saved for each control in the DLL (this will be a REVIT standard), and set it to the sErrMsg that is located in your application control.

3.1.5 Reporting of Error

The reporting of the error should be done at the highest level. What is meant by highest level? Within an application, there are events that are invoked when a menu is selected or a button is pressed. It is at this level that the error be reported. The error can be reported in a number of manners. In the simplest form, a message box with the error is shown to the user. Another method would be to make use of a notes listbox that gets filled in with the details of the processing. A third method is reporting the error to a system error log. One or more of these methods can be used. The implementation will depend upon how the application is used. This will be decided during the design time of the application.

NOTE: There might be instances where you want to report at a lower level (this will be up to the designer/engineer on an as needed basis).

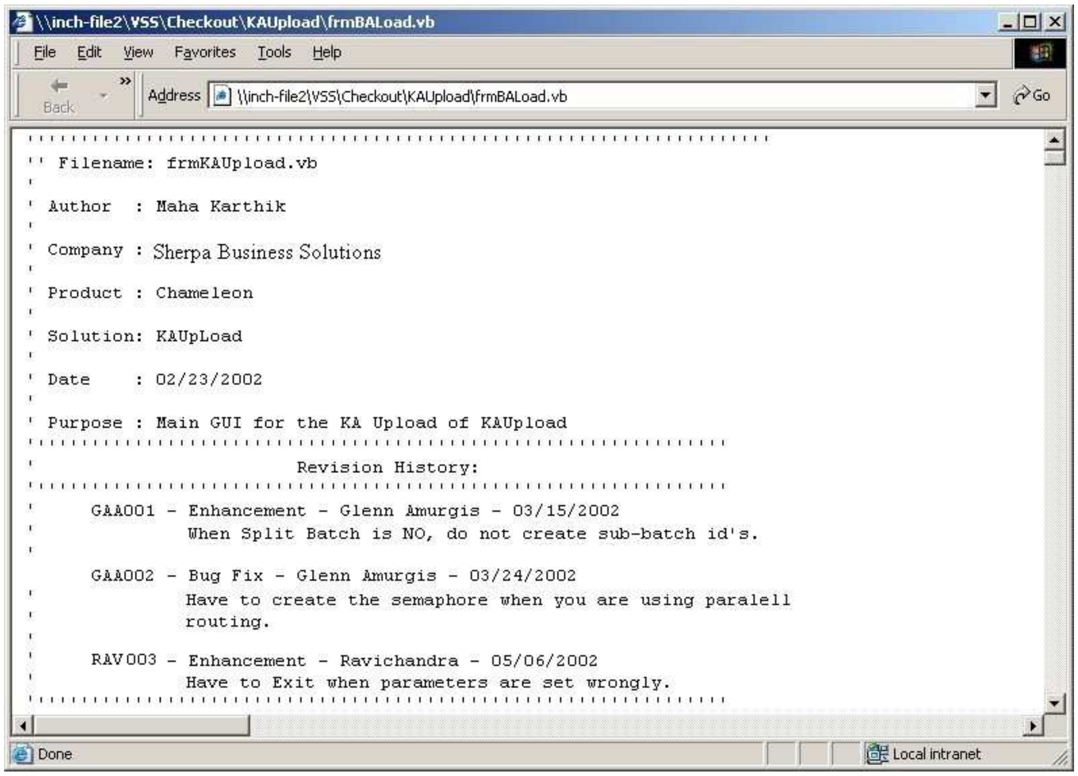
3.2 Sub Routines

Subroutines should be used only when needed/required. For example, the object constructor “NEW” is a sub so you cannot avoid using a subroutine. We will standardize on functions. NOTE: callbacks to form elements (e.g. push button or menu) make use of subroutines

4 DOCUMENTATION

4.1 FILE HEADER

Each file must have a file documentation header. The figure below shows what this file header should look like


\\inch-file2\VSS\Checkout\KAUpload\frmBALoad.vb
File Edit View Favorites Tools Help
Back
Address \\inch-file2\VSS\Checkout\KAUpload\frmBALoad.vb
Go
Done
Local intranet

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'' Filename: frmKAUpload.vb
'
' Author   : Maha Karthik
'
' Company  : Sherpa Business Solutions
'
' Product  : Chameleon
'
' Solution: KAUpLoad
'
' Date     : 02/23/2002
'
' Purpose  : Main GUI for the KA Upload of KAUpload
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'                        Revision History:
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'     GAA001 - Enhancement - Glenn Amurgis - 03/15/2002
'              When Split Batch is NO, do not create sub-batch id's.
'
      GAA002 - Bug Fix - Glenn Amurgis - 03/24/2002
'              Have to create the semaphore when you are using paralell
'              routing.
'
'     RAV003 - Enhancement - Ravichandra - 05/06/2002
'              Have to Exit when parameters are set wrongly.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

The table below describes all of the items that need to be present in the file header comment.

Item	Description
Filename	Name of the source file
Author	Name of the Author
Company	Name of the Company
Product	Name of the Product to which the solution belongs. This will be given by the specification writer
Solution	Name of the solution to which the source belongs
Date	Date which the source was created
Purpose	A brief description of what the source does
Revision History	There needs to be a revision history block. This block will be used to tell what changes where made to the file over time, why were they made, who did the changes, and when did they change them

4.2 IN-LINE

The amount of inline documentation that is needed is subjective. It is the author's feeling that well written code is self documenting.

However, there will be times when in-line documentation is needed. Some basic rules of in-line documentation is defined below.

4.2.1 Complex Function

One case for in-line documentation is when a function is   “complex” or “busy”. Create

In line documentation is useful when there are a number of steps/sub steps being performed in a given function/subroutine. Create a documentation block at the start of the “complex” or busy part.

Example:

```
Public Sub MnuProcessBatch_Click(ByVal sender As System.Object,
           ByVal e As System.EventArgs) Handles MnuProcessBatch.Click
Dim rc As Int32
Dim zipfile, batchname As String
Dim afileentries As String()
Dim i As Int32

   frmSplashBAUPLD.Visible = True
   frmSplashBAUPLD.lblText.Text = "Updating Data, Please WAIT !!!"
   frmSplashBAUPLD.Update()

   Cursor.Current = System.Windows.Forms.Cursors.WaitCursor
   Cursor.Show()

   rc = 0 : i = 0
   zipfile = ""
   LbMsg.Items.Clear()
   LbMsg.Items.Add("Starting Processing, please wait ...")

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Basic Logic:
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Step 1: Clear working area
' Step 2: Extract Zip (If Zip is Encrypted, first DeEncrypt) to working
          area.
' Step 3: Build Batch Area in wip and write control file there.
' Step 4: Build Sub-batch area, NOTE: If General.SplitBatch is
'         This will be build in the "Parent" Batch - one in the same
' Step 5: Update the Workflow, wil vary depending w/not ‘
'         General.SplitBatch is set to Yes.
' Step 6: Finish the .zip file (either remove or rename).
' Step 7: Re-Display the values in the GUI
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
   gErrMsg = ""
   rc = BuildAndClearTemp()
   If rc = 0 Then
      If (lbInBatch.SelectedItems.Count = 0) Then
          LbMsg.Items.Add("Must First Select a Batch to Work On ...")
          frmSplashBAUPLD.Visible = False
          Exit Sub
      Else
          zipfile = CStr(lbInBatch.SelectedItems(i))
          afileentries = Split(zipfile, ".")
          Update()
          batchname = afileentries(0)
          LbMsg.Items.Add("   1.) Unzipping Data")
          If gZipEncrypted = "YES" Then
              rc = DecryptZip(zipfile, batchname)
              If rc = 0 Then
                 gZipObj.ZipFilename = gWorkDir & "\" & batchname & "." &
gZipExt
                 rc = ExtractZip(zipfile, batchname)
              End If
          Else
              gZipObj.ZipFilename = gInputDir & "\" & zipfile
              rc = ExtractZip(zipfile, batchname)
          End If

          If (rc = 0) Then
             If gOutputZipEncrypted = "YES" Then
                rc = EncryptZip(zipfile, batchname)
                If rc = 0 Then
                   File.Delete(gWorkDir & "\" & batchname & "." & gZipExt)
                End If
             End If
         End If
      End If
   End If

   If rc = 0 Then
      rc = LoadBatchControl(batchname)
      If rc = 0 Then
         rc = ParentDirectory(batchname)
         If rc = 0 Then
            LbMsg.Items.Add("   2.) Successfully Created Batch Dir")
            rc = SubDirectory(batchname)
            If rc = 0 Then
               If (gSplitBatch = "YES") Then
                   rc = UpdateDatabaseSplit()
               Else
                   rc = UpdateDatabaseNorm()
               End If
            End If
            If (rc = 0) Then
               LbMsg.Items.Add("   7.) Updated Database")
               Update()
            End If

            If (rc = 0) Then
               If gBackupZip = "YES" Then
                     rc = BackupZip(zipfile)
                     If rc = 0 Then
                        LbMsg.Items.Add("x.) The ZipFile is backed up")
                     End If
               Else
                   IO.File.Delete(zipfile)
               End If
               rc = UpdateDisplay(zipfile)
               If rc = 0 Then
                  File.Create(gBaseDir & "\" & gWipDir & "\" & batchname &
"\upload.sem")
               End If
            End If
        End If
     End If
     zipfile = ""
     i = 0
     frmSplashBAUPLD.Visible = False

     Cursor.Current = System.Windows.Forms.Cursors.Default
     If (rc > 0) Then
        If (Len(gErrMsg) > 0) Then
           LbMsg.Items.Add(gErrMsg)
           MsgBox(gErrMsg)
        End If
     End If
End Sub
```

4.2.2 Do not Document the Obvious

Some developers get overboard with in-line documentation.
Do not document the obvious

e.g.

Dim rc as int32

 ‘ Setting rc to zero
  rc = 0
.

4.2.3 Making sure Inline Matches Logic

If there is in-line documentation and you are changing the logic, make sure you
Update the in-line documentation. There is NOTHING WORSE than BAD documentation. It is misleading and will cause others that work on the source to distrust any documentation.

4.3 Revision(s)

When there is a revision, we need to add a comment in two places.

The first area that we have to document is the File level comment. In this comment there is a revision block. After the first release, anytime an engineer changes the source, the change notice should be added here. The syntax for this block is as follows:

‘ REFID – TYPE – Engineer – Date
‘   Description

Where:

REFID is a Unique Id that will be used to Cross-Reference an In-Line comment on where the changes took place. The format for REFID will be AAANNN: (E.g. GAA001)

- **AAA** – 3 unique characters that identify the engineer. This should be standard and not change at any point of time for a particular engineer. New engineers or those with similar names should ensure that there is no repetition of this ID.

- **NNN** – Running sequence number which tracks the number of changes to the application. This should not be a sequence number specific to each engineer but should be continuous across all changes made to the application.
  E.g. GAA001

RAV002
GAA003
HAR004
GAA005

Note:
Since the current process has the sequence no. specific to each engineer, for all future enhancement follow the new procedure by counting the no of changes made and take the next no. in the sequence.

i.e. if there are a total no of 5 changes made, take the next sequence no as 006. So the REFID will read HAR006.

TYPE – the type of fix either Enhancement or bug-fix

Engineer – The name of the engineer who made the corrections

Date – The date the changes were made in mm/dd/yyyy format

Description: A brief description of what was corrected

Example:

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'                        Revision History:
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'GAA001 - Enhancement - Glenn Amurgis - 03/15/2002
'         When Split Batch is NO, do not create sub-batch id's.

'GAA002 - Bug Fix - Glenn Amurgis - 03/24/2002
'         Have to create the semaphore when you are using paralell
'         routing.

'RAV003 - Enhancement – Ravi Chandra - 05/06/2002
'         Have to exit when wrong parameters are set.

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

The second comment is an inline comment. This should be given for a single line change as well as for changes made to a block of code.

For single line code, just give the comment at the end of line

e.g.

```
Public gValTmpl As String           ' Profile Parm: AppKey.ValTmpl
```

For a block which is changed, this should be placed at the start and end of the change with the reference that was used in the File Level Comment.
e.g.

```
'GAA001 – START
```

Block of Changed Code

```
'GAA001 – END
```

Why do we want it like this. We can look at the start of the file to see the history. We can look at the REFID to find out where in the source it occurred.

4.4 Properties

Each Public Property in a class should have a simple inline comment. It should either state what the property is for and/or how it is derived.

e.g.

```
Public gValTmpl As String            ' Profile Parm: AppKey.ValTmpl
Public gZipExt As String             ' Profile Parm: AppKey.ZipExt


Public gCustId As String             ' CmdLine Parm: First Parm, Customer
Public gProjId As String             ' CmdLine Parm: 2nd Parm, Project
Public gInstallBase As String        ' CmdLine Parm: 3rd Parm, Install Base
```

There are some common comment formats that can be used. The use of these is optional. The advantage of using is it will make a person who is not familiar with the application have a good feel for how/where properties are set.

Comment Comments for Properties

Comment Property Comment	**Description**
Profile Parm: AppKey.XXXXX	This tells that the Property is set from the Profile. XXXXX is the name of the Profile parameter AppKey means the Parameter can be found in the Application's Key block
Profile Parm: General.XXXXX	This tells that the Property is set from the Profile. XXXXX is the name of the Profile parameter General means the parameter can be found in the "General" Block.
CmdLine: Nth Param, Defintion	CmdLine tells that this property was set via a command line parameter. Nth parameter tells which/how it was passed from the command line and Definition is a brief description of what the property is.
Derived = EEEEE	Derived means that value was derived from one or more other values. EEEEE is the expression
Definition	A simple description for working properties. Explains the use of the property.

4.5 Public Methods for DLL

Each public Method for a control that is used by more than one solution must be documented. There should be a comment box that contains the following:

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
' Method Name:
' Purpose    :
' Parameters :
' Returns    :
' Notes      :
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

Where:

Method Name is the name of the method
Purpose is a brief description of the method
Parameters is the parameters that are passed in. Need to tell if byref or by value
Return is the returns
Notes are any notes or special logic that is useful for the caller.

5 Compiling Options

5.1 Release

When building for production, you have to build using the release option

5.2 Strict

When building for production, you have to use the strict option. This forces a very stringent version of the complier and ensures that you are placing proper syntax.

6 Programming/Logic Standards

6.1 Looping Structures

When building a looping structure, make the condition determine what drops out rather than using some goto or other non-structured branch.

e.g. GOOD logic

```
i= 0
do while (i < 10 )
        .
        .
        .
   i = i + 1
end while
```

e.g. BAD Logic

```
i= 0
do while (true )
        .
        .
        .
   i = i + 1
  if ( i >= 10 ) then
    exit while
  end if
end while
```

6.2 GoTo

Do not use

6.3 Exit function or Exit sub

Limit use, there are cases when useful but should be held to a minimum

7 Amount of Nesting/ Amount of Code

Try not to get too crazy on the nesting, Anything more than 4/5 levels tends to be unreadable and unmaintainable. If need be, learn to create functions modules when need to break long and unwieldy nested loops within a function.

8 ASSEMBLY INFORMATION

In each solution there is an AssemblyInfo.vb. This contains meta data information about your application. An example of this file is as follows:

```
Imports System.Reflection
Imports System.Runtime.InteropServices

' General Information about an assembly is controlled through the following
' set of attributes. Change these attribute values to modify the
information
' associated with an assembly.

' Review the values of the assembly attributes

<Assembly: AssemblyTitle("BAUpload")>
<Assembly: AssemblyDescription("Application to Upload Batches to BA
system")>
<Assembly: AssemblyCompany("Sherpa Business solutions ")>
<Assembly: AssemblyProduct("Chameleon")>
<Assembly: AssemblyCopyright("Copyright (C) Sherpa Business Solutions - All
Rights Reserved – mm/dd/yyyy")>
<Assembly: AssemblyTrademark("")>
<Assembly: CLSCompliant(True)>

'The following GUID is for the ID of the typelib if this project is exposed
to COM
<Assembly: Guid("08C1F0D1-9506-4EC1-AE44-0860DBF02761")>

' Version information for an assembly consists of the following four
values:
'
'       Major Version
'       Minor Version
'       Build Number
'       Revision
'
' You can specify all the values or you can default the Build and Revision
Numbers
' by using the '*' as shown below:

<Assembly: AssemblyVersion("1.0.*")>
```

Here is how to fill it out:

Value	**What to set it to**
`AssemblyTitle`	Name of the application – Designer will tell as part of the specs
`AssemblyDescription`	A brief description on what the application does. Use the intro from the specs
`AssemblyCompany`	Sherpa Business Solutions
`AssemblyProduct`	Chameleon – for the generic workflow apps ChameleonForms – For forms applications
`AssemblyCopyright`	Copyright © Sherpa Business Solutions – All Rights Reserved – 02/15/2002
`AssemblyTrademark`	To be determined – leave blank for now
`CLSCompliant`	TRUE
`AssemblyGuid`	Let system generate
`AssemblyVersion`	Let system generate

NOTE: This information is available when examining the file via exployer.

Annexure A

Information Classification Details

Classification: Firstsource RESTRICTED
Information Owner (IO): DGM (Software)
Information Custodian (IC): Software Team
Authorization List (AL): Software Team
Declassify on: NEVER

Annexure B

Changes since version 1.0

1. Version 1.0 Released on 28-Apr-05
2. Version 2.0 Released on 5-Jan-2010