Contents

The following Help Topics are available:

Reusable Modules

Reusable Forms

For Help on Help, Press F1

Initialization File Functions

InitIni - Initialize initialization file parameters
WriteToIni - Write a value to the initialization file
GetFromIni - Retrieve a value from an initialization file

InitIni

Initializes the ini file module

Syntax

InitIni(*sApplicationName*, *sIniFileName*)

Remarks

This subroutine is called once during program start up to set the names of the application and the ini file.

The InitIni uses the following parts:

sApplicationName	String expression containing the application name.
sIniFileName	String expression containing the ini file name

Example

InitIni "MIS Application", "misapp.ini"

Module: RUFINI.BAS

WriteToIni

Writes a value to the applciation's inialization file

Syntax

WriteToIni (*sKey*, *sValue*)

Remarks
Use this subroutine to store ini file values.

The WriteToIni uses the following parts:

sKey String expression containing the key for identifing the value

sValue String expressioncontaining the value to be saved

Examples

WriteToIni "DatabasePath", *"c:\data\db.mdb"*

WriteToIni "Xcord", *"55"*

Module: RUFINI.BAS

GetFromIni

Returns a string from the application's initialzation file.

Syntax

sValue = **GetFromIni** (*sKey*, *nSize)*

Remarks
Use this function to retrieve a value in the form of a string from an initialization file.

The GetFromIni uses the following parts:

sKey	String expression containing the key for identifing the value
nSize	Numeric expression containing the maximun size of the return string

Return Value

sValue The value in a string expression from the initialization file.

Example

```
Dim sPath as string

sPath = GetFromIni ("DatabasePath", 100)
```

Module: RUFINI.BAS

Message Box Functions

InitMB - Initlizes the message box module
AskUser - Prompts the user for a confirmation
InformUser - Displays a message with an exclamation icon
StopUser - Displays a message with a stop icon

InitMB

Set the application name in the message box module

Syntax

InitMB *sApplicationName*

Remarks
Call this subroutine at program start up to set the name of the application. This name will appear in the title bar of the message box.

sApplicationName String expression containing the application name

Example

InitMB MainForm.caption

Module: RUFMB.BAS

AskUser

Asks the user to answer "Yes" or "No" to a message box prompt. Uses the question mark icon and sounds a beep.

Syntax

AskUser (*sQuestion*)

Remarks
Use this function to ask the user to confirm a action.

sQuestion	String expression containing the message in the form of a question.

Return Value

bResponse	Boolean expression: True for a "Yes" selection; False for "No".

Example

```
Dim bResponse as integer

If AskUser ("Do you want to exit?") then
    end
endif
```

Module: RUFMB.BAS

InformUser

Displays a message box with the supplied message with the exclamation icon and an "OK" button and sounds a beep.

Syntax

InformUser *sMsg*

Remarks
Use the subroutine to show a message to the user

sMsg String expression containing the message.

Example

InformUser "The record can not be found!"

Module: RUFMB.BAS

StopUser

Displays a message box with the supplied message with the stop icon and an "OK" button and sounds a beep.

Syntax

StopUser *sMsg*

Remarks
Use the subroutine to show a message to the user

sMsg String expression containing the message.

Example

StopUser "Fatal error!"

Module: RUFMB.BAS

Help Functions

InitHelp - Initializes the help function module.
CallHelp - Calls help for a particular topic.
EndHelp - Closes a help file associated with the current application.
HelpContents - Calls the help contents for the current help file.
HelpIndex - Calls the help index for the current help file.
HelpSearch

InitHelp

Initializes the help function module.

Syntax

InitHelp *sFile*, *hMainWnd*

Remarks
Use InitHelp at program start up to set the name of the help file and the progam's handel.

The **InitHelp** uses the following parts:

sFile	String expression containing the name of the help file.
hMainWnd	Handel (integer expression) of the main form of the application.

Example

InitHelp "myapp.hlp", MainForm.hWnd

Module: RUFHELP.BAS

CallHelp

Calls the application's help file and the topic for the supplied context ID.

Syntax

CallHelp *lContext*

Remarks
Place CallHelp in a form's KeyDown event and set the form's KeyPreview property to True. When the F1 key is pressed you can trap the key press and call the topic context ID.

lContext Long expression of the context ID.

Example

```
Const CONTEXT_ID& = 1234

Sub Form_KeyDown (KeyCode As Integer, Shift As Integer)
    If KeyCode = KEY_F1 Then
        CallHelp CONTEXT_ID
    End If
End Sub
```

Module: RUFHELP.BAS

EndHelp

End WinHelp when you close your application.

Syntax

EndHelp

Remarks
Call EndHelp when you close your application; if you help file is still running this subroutine will close it for you.

Example

```
Sub Form_Unload (Cancel As Integer)
    EndHelp
    End
Exit Sub
```

Module: RUFHELP.BAS

HelpContents

Calls the help content of your applications help file.

Syntax

HelpContents

Remarks
Call this subroutine to view the contect topic of your help file.

Example

```
Sub helpMenu_Click (Index As Integer)

    Select Case Index

        Case 1 'contents
            HelpContents

        Case 2 'index
            HelpIndex

    End Select

End Sub
```

Module: RUFHELP.BAS

HelpIndex

Calls the help index of your applications help file.

Syntax

HelpIndex

Remarks
Call this subroutine to view the index topic of your help file.

Example

```
Sub helpMenu_Click (Index As Integer)

    Select Case Index

        Case 1 'contents
            HelpContents

        Case 2 'index
            HelpIndex

    End Select

End Sub
```

Module: RUFHELP.BAS

General Functions

AddNewItem - Adds a string and an ID (to the ItemData property) to a list or combo box.
ArrowCursor - Changes the cursor to an arrow.
CenterFromScreen - Centers a form on a screen.
CenterIt - Centers a form on top of another form.
CheckCmdLine - Check for the presents of a command line switch.
Encrypt - Encrypts or decripts a string of characters.
FillFullScreen - Resizes a form to fill the whole screen without maximizing the form.
FindProgram - Determines if another instance of the same program is already running.
GetCmdLineStr - Retrieves a parameter from a command line switch.
HourglassCursor - Changes the mouse pointer to an hourglass.
ModalForm - Centers and displays a form in a modal fashion.
ModelessForm - Centers and displays a form in a modaless fashion.
SelectText - Highlight the entire text in an edit control.
ShowYesNo - Converts boolean values of True or False to "Yes" or "No" strings.

AddNewItem

Adds a new item to a list or combo box along with a ItemData value.

Syntax

AddNewItem *ctrlControl, sItem, lItemIndex*

Remarks
Use this subroutine when filling a list with items that have accompaning ID numbers

The AddNewItem uses the following parts:

ctrlControl	List or combo box control.
sItem	String expression of the item to add to the list.
lItemIndex	Long number ID associated with the item added to the list.

Example

AddNewItem "September", 9
AddNewItem "October", 10
AddNewItem "November", 11
AddNewItem "December", 12

Module: RUFUTIL.BAS

ArrowCursor

Changes the mouse cursor to an arrow.

Syntax

ArrowCursor

Example

ArrowCursor

Module: RUFUTIL.BAS

CenterFromScreen

Centers a form based on current screen dimentions.

Syntax

CenterFromScreen *frmMyForm*

Remarks
Use this subroutine to center a form prior to displaying it. Subroutines ModalForm and ModelessForm both use CenterFromScreen.

frmMyForm Form to be centered.

Example

```
Load frmNew
CenterFromScreen frmNew
frmNew.Show 1
```

Module: RUFUTIL.BAS

CenterIt

Centers one form on top of another form.

Syntax

CenterIt *frmOld, frmNew*

Remarks
Use this subroutine to center a form prior to displaying it.

The CenterIt uses the following parts:

frmOld Form currently displayed.
frmNew Form about to be displayed.

Example

```
Load SubForm
CenterIt Mainform, SubForm
SubForm.Show 1
```

Module: RUFUTIL.BAS

CheckCmdLine

Checks the comand line for the presence of a comand line switch.

Syntax

bPresent = **CheckCmdLine** *sPram*

Remarks
Use CheckCmdLine to deterime what parameters have been set. The function ignores case.

sPram	String expression containing the parameter to check for.

Return Value

bPresent	Boolean expression: True - string found; False - string not found.

Example

```
If Not CheckCmdLine "nologo" Then
    AboutBox
EndIf
```

Module: RUFUTIL.BAS

Encrypt

Encrypts or decripts a string.

Syntax

sEncryptStr = **Encrypt** *sString*

Remarks
Use Encrypt to encode a password. Passing an encrypted string to Encrypt returns the unencrypted string.

sString	String expression to encrypt or decript.

Return Value

sEncryptStr	Encrypted or decripted string expression.

Example

```
sPassword = Encrypt "password"
```

Module: RUFUTIL.BAS

FillFullScreen

Resizes a form to fill the full screen withour maximizing it.

Syntax

FillFullScreen *frmMyForm*

Remarks
Use FillFullScreen when you don't want to display the form fully maximized.

frmMyForm Form to resize.

Example

FillFullScreen MainForm

Module: RUFUTIL.BAS

FindProgram

Determines if another instance of the same program is already running and ends the current program if a another instance is found.

Syntax

FindProgram *sCaption*

Remarks
Use FindProgram during program startup to allow only one instance to run at a time. In order for **FindProgram** to work, you must not set the caption of your main form until after you call **FindProgram**. Otherwise **FindProgram** will find the current instance of the program an not a previous running instance.

sCaption String expression containing the caption of the programs main form.

Example

```
MainForm.Caption = ""    'should be blank
FindProgram TheAppTitle
MainForm.Caption = TheAppTitle
```

Module: RUFUTIL.BAS

HourglassCursor

Changes the mouse cursor to an hourglass.

Syntax

HourglassCursor

Example

HourglassCursor

Module: RUFUTIL.BAS

ModalForm

Loads, centers and displays a modal form.

Syntax

ModalForm *frmNew*

Remarks

frmNew Form to display.

Example

ModalForm MyForm

Module: RUFUTIL.BAS

ModelessForm

Loads, centers and displays a modaless form.

Syntax

ModelessForm *frmNew*

Remarks

frmNew Form to display.

Example

ModelessForm MyForm

Module: RUFUTIL.BAS

SelectText

Select or highlights all the in an edit control, mask edit, etc...

Syntax

SelectText *ctrlText*

Remarks
As one tabs around a Windows data input screen the text in each text control is highlighted when it receives the focus. But for this behavior to happen the text must be "selected" when (1)the program populates the fields with data and (2) a field loses focus.

ctrlText Control containing text.

Example

SelectText txtAddress

Module: RUFUTIL.BAS

ShowYesNo

Converts boolean values True and False to "Yes" or "No".

Syntax

sStr **ShowYesNo** *bVal*

Remarks

bVal Boolean expression.

Return Value

sStr String expression.

Example

```
sStr = ShowYesNo (True)          ' sStr = "Yes"
```

Module: RUFUTIL.BAS

GetCmdLineStr

Returns a parameter string from the command line.

Syntax

sParameter = **GetCmdLineStr** *sCmdSwitch*

Remarks
Use this function to retrieve a user supplied parameter from the command line. Command line parameters must follow this syntax: <Switch>:<ParameterString>. The colon is required. GetCmdLineString will ignore the case of characters.

sCmdSwitch String expression of the switch to search for.

Return Value

sParameter String expression containing the user supplied parameter.

Example

```
'command line: myapp.exe ini:myapp2.ini
'check for alternate .ini file
sParameter = GetCmdLineStr "ini"
If strcomp(sParameter, "" ) = 0 Then
      sIniFile = sParameter
Else
      sIniFile = "myapp.ini"
EndIF
```

Module: RUFUTIL.BAS

Check State Function

Validates a two character state abbreviation.

Syntax

bOK = **CheckState** (*sState*)

Remarks
CheckState check for valid state abbreviation including Proto Rico and the Virgin Islands.

sState String expression of a two character state abbreviation in **upper case**.

Example

```
bOK = CheckState ("GA")      'bOK = True
```

Module: RUFSTATE.BAS

MAPI Functions

SendSingleMailMsg - Send a Single Mail Message
InitMultiMapi - Initialize the MAPI module for a mutli-mailing session.
ConstructMultiMailMsg - Send one message during a mutli-mailing session.
MapiSignOff - End the current MAPI session.

SendSingleMailMsg

Initiate a MAPI session, sends a message and closes the session.

Syntax

SendSingleMailMsg *frmMapi, sSubject, sMsg, sName*

Remarks
The subroutine simplies the process of sending a email message.

The SendSingleMailMsg uses the following parts:

frmMapi	Form containing the MAPI controls
sSubject	String expression containing the subject of the message.
sMsg	String expression containing the e-mail message.
sName	String expression containing the name of the message recipient.

Example

SendSingleMailMsg MailForm, "Attention", "Your work is overdue!", "Joe Cool"

Module: RUFMAIL.BAS

InitMultiMapi

Initializes the MAPI module for sending several message during a MAPI session.

Syntax

bOK = **InitMultiMapi** (*frmMail*)

Remarks
Use InitMultiMapi along with ConstructMultiMailMsg and MapiSignOff when sending out more than one e-mail message per session.

frmMail	Form containing the MAPI controls.

Return Value

bOK	Boolean expression; True if the session startup was sucessful.

Example

```
If InitMultiMapi (MailForm) Then
    ConstructMultiMailMsg (sSubject, sMessage, "Joe Cool")
    ConstructMultiMailMsg (sSubject, sMessage, "Joe Cool, Jr.")
    MapiSignOff
EndIf
```

Module: RUFMAIL.BAS

ConstructMultiMailMsg

Constructs and send a message during a multi-message MAPI session.

Syntax

ConstructMultiMailMsg (*sSubject, sMsg, sName*)

Remarks
The MAPI session must be initialized first by using InitMultiMapi and concluded with MapiSignOff.

The ConstructMultiMailMsg uses the following parts:

sSubject	String expression containing the subject of the message.
sMsg	String expression containing the e-mail message.
sName	String expression containing the name of the message recipient.

Return Value

bOK	Boolean expression; True if message sent sucessfully

Example

```
If InitMultiMapi (MailForm) Then
    ConstructMultiMailMsg (sSubject, sMessage, "Joe Cool")
    ConstructMultiMailMsg (sSubject, sMessage, "Joe Cool, Jr.")
    MapiSignOff
EndIf
```

Module: RUFMAIL.BAS

MapiSignOff

Closes the current MAPI session.

Syntax

MapiSignOff

Remarks
Use MapiSignOff after sending multi e-mail messages.

Example

```
If InitMultiMapi (MailForm) Then
    ConstructMultiMailMsg (sSubject, sMessage, "Joe Cool")
    ConstructMultiMailMsg (sSubject, sMessage, "Joe Cool, Jr.")
    MapiSignOff
EndIf
```

Module: RUFMAIL.BAS

Date/Time Validation Functions

ValidateDate - Validates a date string.
ValidateTime - Validates a time string.

ValidateDate

Determines if a string in a mask edit control contains a valid date.

Syntax

bOK = **ValidateDate (***ctrlMaskEdit*)

Remarks
The date format is mm/dd/yyyy.

ctrlMaskEdit MaskEdit control that contains a date string

Return Value

bOK Boolean expression; True if the date is valid.

Example

```
If Not ValidateDate (txtMaskEdit) then
    Beep
EndIf
```

Module: RUFTIME.BAS

ValidateTime

Determines is a string contains a valid 12 hour format time.

Syntax

bOK = **ValidateTime** (*sTime*)

Remarks
The time format expected is HH:MM AM/PM.

sTime	String contain a time in 12 hour format.

Return Value

bOK	Boolean expression; True if the date is valid.

Example

```
If Not ValidateTime (sTime) then
    Beep
EndIf
```

Module: RUFTIME.BAS

Database Functions

AddQuote - Addes a single quote mark where one has been removed.
AddQuoteV - Addes a single quote mark where one has been removed, uses a variant parameter.
AddToInsert - Adds a value to an insert statement.
AddToUpdate - Adds a value to an update statement.
AddToUpdateV - Adds a value to an update statement, uses a variant parameter.
CheckAndSaveCbo - Adds new items from a combo box to a list stored in a database table.
CompactDB - Compacts a database.
CreateInsert - Starts a new insert statement.
CreateUpdate - Starts a new create statement.
DatabaseError - Displays a database error message.
ExecuteInsert - Executes an SQL statement.
GetCBOID - Returns the item data value for a combo box selection.
GetID - Returns a new ID number.
GetInsertStatement - Returns a complete SQL insert statement.
GetLBID - Returns the item data value for a list box selection.
GetUpdateStatement - Returns a complete SQL update statement.
KeyFound - Determines is a key value already exists in the database
LoadCombo - Fills a combo box with database records.
LoadListBox - Fills a list box with database records.
RemoveQuote - Remove a single quote from a string and replaces it with another character.
ScanCombo - Scans a combo box for a unique item data value.
ScanListBox - Scans a list box for a unique item data value.
ScanMultiListBox - Scans a multi select list box for a unique item data value.
SetSystemDB - Set the name of the system table that maitains available ID numbers.

AddQuote

Inserts single quote marks into a string that had the remove previously.

Syntax

sNewString = **AddQuote**(*sOldString*)

Remarks
Use AddQuote to insert single quote marks that have been removed during the process of adding records to a database through the AddToUpdate or AddToInsert or RemoreQuote functions. These functions replace the single quote with the backward quote mark (`).

sOldString String expression.

Return Value

sNewString String expression.

Example

```
Dim sTmp$
sTmp = dsData("LastName")
sTmp = AddQuote(sTmp)
```

Module: RUFDB.BAS Database functions

AddQuoteV

Inserts single quote marks into a string that had the remove previously. This version of the function takes a variant rather then a string parameter.

Syntax

sNewString = **AddQuoteV**(*vOldString*)

Remarks
Use AddQuote to insert single quote marks that have been removed during the process of adding records to a database through the AddToUpdate or AddToInsert or RemoreQuote functions. These functions replace the single quote with the backward quote mark (`).

vOldString Variant expression, such as a dynaset or snaphot record set member.

Return Value

sNewString String expression.

Example

sTmp = **AddQuoteV**(dsData("LastName"))

Module: RUFDB.BAS Database functions

AddToInsert

Adds a value an insert statement under construction.

Syntax

AddToInsert *sValue*, *bLit*

Remarks
Insert statements can be contrusted by using CreateInsert, **AddToInsert** and GetInsertStatement. CreateInsert accepts the name of table to insert to and clears out the insert buffer. For each field in the table **AddToInsert** is called to add a value to the insert statement. Each field must be inserted in the same as the target table's layout. Once all the elements have been added, calling GetInsertStatement returns the completed statement that can now be executed.

The AddToInsert uses the following parts:

sValue String expression containing the key for identifing the value.

bLit Boolean expression; denotes if *sValue* is to be added as a literial value. True if literial (string or date) value; False if non-literial (numeric).

Example

```
Dim sSQL$
CreateInsert "CustomerTable"
AddToInsert "2938", False              'field 1, numeric
AddToInsert "Joe", True                'field 2, text
AddToInsert "Cool", True               'field 3, text
AddToInsert "06/13/1994", True'field 4, date
sSQL = GetInsertStatement()            'sSQL = "INSERT INTO CustomerTable values ( 2938, 'Joe', 'Cool', '06/13/1994');"
```

Module: RUFDB.BAS Database functions

AddToUpdate

Adds a value an update statement under construction.

Syntax

AddToUpdate *sName*, *sValue*, *bLit*

Remarks
Update statements can be contrusted by using CreateUpdate, **AddToUpdate** and GetUpdateStatement. CreateUpdate accepts the name of table to update to and clears out the update buffer. For each field in the table **AddToUpdate** is called to add a value to the update statement. Once all the elements have been added, calling GetUpdateStatement returns the completed statement that can now be executed.

The AddToUpdate uses the following parts:

sName String expression containing the name of the field to update.

sValue String expression containing the key for identifing the value.

bLit Boolean expression; denotes if *sValue* is to be added as a literial value. True if literial (string or date) value; False if non-literial (numeric).

Example

```
Dim sSQL$
CreateUpdate "CustomerTable"
AddToUpdate "FirstName", "Joe", True
AddToUpdate "LastName", "Cool", True
AddToUpdate "LastOrderDate", "06/13/1994", True
sSQL = GetUpdateStatement("where CustomerID = 2938" )        'sSQL = "UPDATE CustomerTable set FirstName = 'Joe', LastName    = 'Cool', LastOrderDate    = '06/13/1994' where CustomerID = 2938;"
```

Module: RUFDB.BAS Database functions

AddToUpdateV

Adds a variant value an update statement under construction.

Syntax

AddToUpdateV*sName*, *sValue*, *bLit*

Remarks
Update statements can be contrusted by using CreateUpdate, **AddToUpdateV** and GetUpdateStatement. CreateUpdate accepts the name of table to update to and clears out the update buffer. For each field in the table **AddToUpdateV** is called to add a value to the update statement. Once all the elements have been added, calling GetUpdateStatement returns the completed statement that can now be executed.

The AddToUpdateV uses the following parts:

sName String expression containing the name of the field to update.

sValue String expression containing the key for identifing the value.

bLit Boolean expression; denotes if *sValue* is to be added as a literial value. True if literial (string or date) value; False if non-literial (numeric).

Example

```
Dim sSQL$
CreateUpdate "CustomerTable"
AddToUpdate "FirstName", "Joe", True
AddToUpdate "LastName", "Cool", True
AddToUpdateV "LastOrderDate", Format(Now, "mm/dd/yyyy"), True
sSQL = GetUpdateStatement("where CustomerID = 2938" )        'sSQL = "UPDATE CustomerTable set FirstName = 'Joe', LastName   = 'Cool', LastOrderDate   = '06/13/1994' where CustomerID = 2938;"
```

Module: RUFDB.BAS Database functions

CompactDB

Compacts an Access ® database

Syntax

CompactDB *sCompDBName*

Remarks

Use CompactDB to optimize a database file.

sCompDBName String expression, name of database file to compact.

Example

CompactDB "sales.mdb"

Module: RUFDB.BAS Database functions

CreateInsert

Start construction of a new insert statement.

Syntax

CreateInsert *sTable*

Remarks
Insert statements can be contrusted by using **CreateInsert**, AddToInsert and GetInsertStatement. **CreateInsert** accepts the name of table to insert to and clears out the insert buffer. For each field in the table AddToInsert is called to add a value to the insert statement. Each field must be inserted in the same as the target table's layout. Once all the elements have been added, calling GetInsertStatement returns the completed statement that can now be executed.

sTable String expression containing the name of the table to insert a record into.

Example

```
Dim sSQL$
CreateInsert "CustomerTable"
AddToInsert "2938", False                'field 1, numeric
AddToInsert "Joe", True          'field 2, text
AddToInsert "Cool", True                 'field 3, text
AddToInsert "06/13/1994", True  'field 4, date
sSQL = GetInsertStatement()              'sSQL = "INSERT INTO CustomerTable values ( 2938, 'Joe', 'Cool', '06/13/1994');"
```

Module: RUFDB.BAS Database functions

CreateUpdate

Start construction of a new insert statement.

Syntax

CreateUpdate *sTable*

Remarks
Update statements can be contrusted by using **CreateUpdate**, AddToUpdate and GetUpdateStatement. **CreateUpdate** accepts the name of table to insert to and clears out the update buffer. For each field in the table AddToUpdate is called to add a value to the insert statement. Each field must be inserted in the same as the target table's layout. Once all the elements have been added, calling GetUpdateStatement returns the completed statement that can now be executed.

sTable String expression containing the name of the table to update.

Example

```
Dim sSQL$
CreateUpdate "CustomerTable"
AddToUpdate "FirstName", "Joe", True
AddToUpdate "LastName", "Cool", True
AddToUpdateV "LastOrderDate", Format(Now, "mm/dd/yyyy"), True
sSQL = GetUpdateStatement("where CustomerID = 2938" )        'sSQL = "UPDATE
CustomerTable set FirstName = 'Joe', LastName   = 'Cool', LastOrderDate   = '06/13/1994' where
CustomerID = 2938;"
```

Module: RUFDB.BAS Database functions

DatabaseError

Reports a database error.

Syntax

DatabaseError

Remarks
Use Database error to reports error that may take place during a database open procedure.

Example

```
On Error Goto dbErr

Set TheDatabase = OpenDatabase(sDatabasePath)
exit sub

dbErr:
DatabaseError
Exit Sub
```

Module: RUFDB.BAS Database functions

ExecuteInsert

Executes an SQL insert or update statement.

Syntax

ExecuteInsert *sBuff*

Remarks
Using this procedure will centralize all excecute commands into one module. This make it easy to modify your program to use pass through queries without making major modifcations to your program.

sBuff String expression containing the an update, insert or delete statement.

Example

```
Dim sBuff$
sBuff = "Delete from Customer where CustID = 7"
ExecuteInsert sBuff
```

Module: RUFDB.BAS Database functions

GetCBOID

Returns the ItemData value for the item currently selected in combo box.

Syntax

lID = **GetCBOID** (*cboCtrl*, *sField*)

Remarks
GetCBOID check for a selection in the combo and will display an error message if no selection has been made.

The **GetCBOID** uses the following parts:

cboCtrl Combo box control.

sField String expression name of the item selected.

Return Value

lID Long expression containing the itemdata value.

Example

```
sub cmdSave_click
    Dim lID

    lID = GetCBOIDcboMonth, "Month")
exit sub
```

Module: RUFDB.BAS Database functions

GetID

Returns a new, unique ID number for adding a new record to a keyed table.

Syntax

lNewID = **GetID**(*sTableName*)

Remarks
Several items need to be in place to use **GetID**. (1) You must have a *one record* system table containing all the next-to-use ID numbers. You can give this table any name you wish and you will tell the RUFDB.BAS module the name of this table through the procedure SetSystemDB. (2) The key field for this table must always be RecNo and the its value will be 1. (3) Each field in the system table that contains the ID number will be named after the table that it supports. Thus an next ID number for the "Customer" table will be in the system table in a field called "Customer". When your program needs to add a new customer record, just call GetID with the parameter, "Customer".

The **GetID** uses the following parts:

sTableName	String expression containing the name of the table to contain a new ID number for.

Return Value

lNewID	Long expression containing the new ID number.

Example

```
Dim lNewCustomerID as Long
lNewCustomerID = GetID("Customer")
```

Module: RUFDB.BAS Database functions

GetInsertStatement

Return a completed insert statement.

Syntax

sInsertStatement = **GetInsertStatement**

Remarks
Insert statements can be contrusted by using CreateInsert, AddToInsert and **GetInsertStatement**. CreateInsert accepts the name of table to insert to and clears out the insert buffer. For each field in the table AddToInsert is called to add a value to the insert statement. Each field must be inserted in the same as the target table's layout. Once all the elements have been added, calling **GetInsertStatement** returns the completed statement that can now be executed.

Return Value

sInsertStatement String expression containing an insert statement.

Example

```
Dim sSQL$
CreateInsert "CustomerTable"
AddToInsert "2938", False                'field 1, numeric
AddToInsert "Joe", True          'field 2, text
AddToInsert "Cool", True                 'field 3, text
AddToInsert "06/13/1994", True           'field 4, date
sSQL = GetInsertStatement()              'sSQL = "INSERT INTO CustomerTable values ( 2938,
'Joe', 'Cool', '06/13/1994');"
```

Module: RUFDB.BAS Database functions

GetLBID

Returns the ItemData value for the item currently selected in list box.

Syntax

lID = **GetLBID**(*lstCtrl*, *sField*)

Remarks
GetLBID check for a selection in the list and will display an error message if no selection has been made.

The **GetLBID** uses the following parts:

lstCtrl List box control.

sField String expression name of the item selected.

Return Value

lID Long expression containing the itemdata value.

Example

```
sub cmdSave_click
    Dim lID

    lID = GetLBID(lstMonth, "Month")
exit sub
```

Module: RUFDB.BAS Database functions

LoadCombo

Fills a combo box with a values from a table. Values are selected and ordered via a query def.

Syntax

LoadCombo *sQDef*, *lDefault*, *cboCtrl*, *bParam*, *sSeparator*, *bClear*

Remarks
Use **LoadCombo** to fill a combo box with items from a database table.

The **LoadCombo** uses the following parts:

sQDef	String expression contain the name of the query def to use. The first field return by the query def must be numberic; it will be inserted into the itemdata property of the list. Also, the query def can accept one parameter with the alias: "Param"
lDefault	Long expression containing the item data value to highlight (if any). If no list box item should be highlighted *lDefault* should be -1.
cboCtrl	Combo box control to fill.
bParam	Boolean expression indicating if the *lDefault* value should be used as a query def parameter. Does not use *lDefault* as a parameter is false.
sSeparator	One character string expression containg a seprator character to place between fields returned by the query def. If no separator is required, use a null string.
bClear	Boolean expression indicating that the combo box should be cleared before filling with data. Clears the combo box if True.

Example

LoadCombo "GetCustomerTypes", 7, cboTypes, False, "", True

Module: RUFDB.BAS Database functions

LoadListBox

Fills a list box with a values from a table. Values are selected and ordered via a query def.

Syntax

LoadListBox(*sQDef, lDefault, lstCtrl, bParam, sSeparator*)

Remarks
Use **LoadListBox** to fill a list box with items from a database table.

The **LoadListBox** uses the following parts:

sQDef	String expression contain the name of the query def to use. The first field return by the query def must be numberic; it will be inserted into the itemdata property of the list. Also, the query def can accept one parameter with the alias: "Param"
lDefault	Long expression containing the item data value to highlight (if any). If no list box item should be highlighted *lDefault* should be -1.
cboCtrl	Combo box control to fill.
bParam	Boolean expression indicating if the *lDefault* value should be used as a query def parameter. Does not use *lDefault* as a parameter is false.
sSeparator	One character string expression containg a seprator character to place between fields returned by the query def. If no separator is required, use a null string.

Example

LoadListBox "GetCustomerTypes", -1, cboTypes, False, ""

Module: RUFDB.BAS Database functions

RemoveQuote

Replaces a single quote mark so that the string can be included in an SQL statement.

Syntax

sNewString = **RemoveQuote**(*sOldString*)

Remarks
Replaces the single quote with the backward quote mark (`).

sOldString String expression.

Return Value

sNewString String expression.

Example

```
Dim sTmp$
sTmp = dsData("LastName")
sTmp = RemoveQuote(sTmp)
```

Module: RUFDB.BAS Database functions

ScanCombo

Searches the itemdata property of a combo box for a matching ID number.

Syntax

nIndex = **ScanCombo**(*lID, cboCtrl*)

Remarks
Use **ScanCombo** to locate and highlight a existing entry in a combo box.

The **ScanCombo** uses the following parts:

lID	Long expression, ID number to search for.
cboCtrl	Combo box control.

Return Value

nIndex	Integer expression. If successful, returns the listindex of the entry. If unsuccessful, returns -1.

Example

```
Dim lCustNo as Long
Dim nIndex as Integer

lCustNo = ssData("CustomerNo")
nIndex = ScanCombo(lCustNo, cboCustomers)
```

Module: RUFDB.BAS Database functions

ScanListBox

Searches the itemdata property of a list box for a matching ID number.

Syntax

nIndex = **ScanListBox**(*lID, lstCtrl*)

Remarks
Use **ScanListBox** to locate and highlight a existing entry in a combo box.

The **ScanListBox** uses the following parts:

lID	Long expression, ID number to search for.
lstCtrl	List box control.

Return Value

nIndex	Integer expression. If successful, returns the listindex of the entry. If unsuccessful, returns -1.

Example

```
Dim lCustNo as Long
Dim nIndex as Integer

lCustNo = ssData("CustomerNo")
nIndex = ScanListBox(lCustNo, cboCustomers)
```

Module: RUFDB.BAS Database functions

ScanMultiListBox

Searches the itemdata property of a multi-select list box for a matching ID number.

Syntax

nIndex = **ScanMultiListBox**(*lID, lstCtrl*)

Remarks
Use **ScanMultiListBox** to locate and highlight a existing entry in a combo box.

The **ScanMultiListBox** uses the following parts:

lID	Long expression, ID number to search for.
lstCtrl	List box control.

Return Value

nIndex	Integer expression. If successful, returns the listindex of the entry. If unsuccessful, returns -1.

Example

```
Dim lCustNo as Long
Dim nIndex as Integer

lCustNo = ssData("CustomerNo")
nIndex = ScanMultiListBox(lCustNo, cboCustomers)
```

Module: RUFDB.BAS Database functions

SetSystemDB

Sets the name of the system table, i.e. the table the contains new record ID numbers.

Syntax

SetSystemDB *sTableName*

Remarks
Call **SetSystemDB** when your program starts up. The system table name will be used by GetID when retrieving a new ID number .

sTableName String expression containing system table name.

Example

SetSystemDB "PTSystem"

Module: RUFDB.BAS Database functions

GetUpdateStatement

Return a completed insert statement.

Syntax

sUpdateStatement = **GetUpdateStatement**(*sWhereClause*)

Remarks
Update statements can be contrusted by using CreateUpdate, AddToUpdate and **GetUpdateStatement**. CreateUpdate accepts the name of table to update and clears out the update buffer. For each field in the table AddToUpdate is called to add a value to the update statement. When calling **GetUpdateStatement** the where clause of the update statement, if any, must be supplied.

sWhereClause String expression containing the where clause of the update statement.

Return Value

sUpdateStatement String expression containing an update statement.

Example

```
Dim sSQL$
CreateUpdate"CustomerTable"
AddToUpdate "FirstName", "Joe", True
AddToUpdate "LastName", "Cool", True
AddToUpdateV "LastOrderDate", Format(Now, "mm/dd/yyyy"), True
sSQL = GetUpdateStatement("where CustomerID = 2938" )       'sSQL = "UPDATE
CustomerTable set FirstName = 'Joe', LastName    = 'Cool', LastOrderDate    = '06/13/1994' where
CustomerID = 2938;"
```

Module: RUFDB.BAS Database functions

PlaySound

Plays a wave file. The windows speaker driver must be installed to use this function.

Syntax

PlaySound(*sWav*)

Remarks

sWav String expression containing the path and name of the wave file.

Example

```
sWav = "c:\wavefile\sound.wav"
PlaySound(sWav)
```

Module: RUFWAVE.BAS

GetErrorMsg

Determines and displays the current Visual Basic error message.

Syntax

GetErrorMsg(*nErr*)

Remarks
Use **GetErrorMsg** in the error handling sections of your subroutines and functions.

nErr Integer expression containing the error number returned by the Err function.

This subroutine requires a table in your application's database named "Error Message" with the following structure:

Field Name	Type	Length
ErrID	Long	4
ErrMsg	String/Text	254

UNIQUE INDEX: PrimaryKey on (ErrID)

This table should contain all valid Visual Basic error codes and descriptions. A copy of this table can be found in the demo program's database file, RUFDEMO.MDB.

Example

```
Sub GenericSub()
    On Error Goto showErr
    ...
    Exit Sub

showErr:
    GetErrorMsg Err
    Exit Sub

End Sub
```

Module: RUFERMSG.BAS

Auxiliary Table Edit Form

Allows editing of a any table with the following structure:

Field Position	Field Type	Field Description
1	Long	ID Number
2	String	Description/Name/Type Field
3	String	Active/Inactive ('1' or '0')

These global variables are declared in RUFAUXED.BAS

sRUFAuxTable$	String expression containing name of the table to edit
lRUFAuxEdHelpID&	Long expression containing the help ID
sRUFAuxIDCaption$	ID caption for RufAuxEdForm
sRUFAuxLable$	Label caption for RufAuxEdForm
sRUFAuxCaption$	Form caption for RufAuxEdForm
sRUFAuxQuery$	Query for loading the RufAuxEdForm list box
sRUFAuxDelCheckQuery$	Query to check for clearance to delete a record
sRUFAuxDelQuery$	Query to delete a record
bRUFAuxDelete%	Boolean value to set the enable property of Delete button
sRUFAuxFields(3) As String	Array contain the field names

Example

```
sRUFAuxTable = "EmpStatus"
lRUFAuxEdHelpID = Auxiluary_Table_Edit_Form
sRUFAuxIDCaption = "Status Number"
sRUFAuxLable = "Employee Status"
sRUFAuxCaption = "Employee Statues"
sRUFAuxQuery = "GetStatusRec"
sRUFAuxDelCheckQuery = "CheckStatusID"
sRUFAuxDelQuery = "DeleteStatusID"
bRUFAuxDelete = True
sRUFAuxLoad = "GetAllStatues"
sRUFAuxFields(0) = "StatusNo"
sRUFAuxFields(1) = "StatusType"
sRUFAuxFields(2) = "Active"
ModalForm RufAuxEdForm
```

Modules: RUFAUXED.FRM, RUFAUXED.BAS

Database Selection Form

The Database Selection Form allows the user the ability to:

View the current database path
Type in a new database path
Display a common file dialog box for selecting a database to open

The current path and name of the database is stored in the sDBPath variable, which is declared in the RUFDB.BAS module. When your application starts up you should load the database name and path from an .ini file and place in into sDBPath. Then call a function such as OpenDB (see the RUFDEMO.FRM in RUFDEMO project code). Another global variable used by this form is bRufDbEnd, a Boolean expression. When bRufDbEnd is True, the program will end when the user presses the Cancel button. If bRufDbEnd is False, Database Selection Form will close and the database that is currently open, if any, will remain unchanged.

Example

```
'get the database & path from the .ini file
sDBPath = GetFromIni("Database", 100)
OpenDB

'Open DB calls the Database Selection Form if an error occurs or sDBPath is blank
...
On Error GoTo dbErr
'stay in a loop until the database is opened
While bDBOK
    bDBOK = False
    bRufDbEnd = True
    ModalForm RUFDBForm
    Hourglasscursor
    Set TheDatabase = OpenDatabase(sDBPath)
    ArrowCursor
Wend

' write the new path
WriteToIni "Database", sDBPath
Exit Sub

dbErr:
    bDBOK = True
    DatabaseError
    Resume Next
...
```

Module: RUFDBFORM.FRM

Login Form

The Login Form provides a handy means to collect the user's name and password for login into your application. These two items are stored in two global variables sUserName and sPassword which are declared in RUFDB.BAS module. After calling the login form check the bLogin variable. The bLogin variable is True when the user pressed the OK button and you can then look up and compare the user's password with the string in sPassword. When bLogin is False, the user has pressed the Cancel button to abort the login.

Example

```
ModalForm RufLogin
If bLogin Then
    While Not CheckPassword() And bLogin
        StopUser "Invalid login!"
        ModalForm RufLogin
        If Not bLogin Then
            End
        End If
    Wend
Else
    End
End If
```

Module: RUFLOGIN.FRM

About Form

With the About Form you do not have to devote development time to showing your company name or copyright information. All you have to do is declare constants for your application title and version number and set a form variable:

In your applications .BAS file declare the following constants:

```
Global Const TheAppTitle$ = "Reusable Functions Demo"
Global Const sVer$ = "1.00" 'version number
```

Then in your main form_load subroutine add this line:

```
Set MainForm = RufMain
```

where *RufMain* is the form the contains your application's icon

Now you can call the About Form in this manner:

```
ModalForm RufAbout
```

Module: RUFABOUT.FRM

HelpSearch

Calls the help content of your applications help file.

Syntax

HelpSearch

Remarks
Call this subroutine to display the WinHelp Search dialog box of your help file.

Example

```
Sub helpMenu_Click (Index As Integer)

    Select Case Index

        Case 1 'contents
            HelpContents

        Case 2 'search
            HelpSearch

    End Select

End Sub
```

Module: RUFHELP.BAS

State Functions

CheckState - Validates a two character state abbreviation
FillStates - Fills a combo box with state abbreviations
FindState - Finds and select a state abbreviations within a combo box filled by FillStates

FillStates

FillStates files a combo box with state abbreviations.

Syntax

FillStates *cboCtrl*

Remarks

You can simplify the entry of a two character state abbreviation by filling up a combo box with a complete list of states rather than letting the user type it in to a text box. When loading an existing record use the subroutine FindState to select a particular state from the combo box.

cboCtrl Combo box control to fill.

Example

```
Sub Form_Load ()
    ...
    FillStates cboStates
    ...
End Sub
```

Module: RUFSTATE.BAS

FindState

Used in conjunction with FillStates, FindStates locates and selects the two character state abbreviation you supplied in the *sInpState* parameter in the combo box denoted in the *cboCtrl* parameter. This combo box was filled with state abbreviations by calling the FillStates subroutine.

Syntax

FindState *sInpState*, *cboCtrl*

Remarks

sInpState String expression containing the state abbreviation

cboCtrl Combo box control containing a list of state abbreviations

Example

FindState "CA", cboStates

Module: RUFSTATE.BAS

CheckAndSaveCbo

If your application maintains a list of values, you may want to store them in a one field, indexed table. By calling CheckAndSaveCbo from the combo box's Lost Focus event you will update your list whenever the user adds a new value.

Syntax

CheckAndSaveCbo *cboCtrl*, *sTable*, *sField*, *bPad*

Remarks
CheckAndSaveCbo always check a new value with its current contents. Only new values are save to the database.

The CheckAndSaveCbo uses the following parts:

cboCtrl	Combo box contain the list of values
sTable	String expression containing the name of the table to store the values. Example table: Rates Fields: RateValues Type: Currency Primary Key: RateValues
sField	String expression containing the field in the table to update
bPad	Boolean expression indication the value should be padded with zeros

Example

```
Sub cboRates_LostFocus()

    CheckAndSaveCbo cboRates, "Rates", "RateValues", True

End Sub
```

Module: RUFDB.BAS Database functions

KeyFound

If you need to determine if a key value already exist in the database prior to saving a new record, call KeyFound to query the table for the new key value.

Syntax

bFound **= KeyFound** (*sTable, sField, sValue*)

Remarks

The KeyFound uses the following parts:

sTable	String expression containing the name of the table to query
sField	String expression containing the name of the field to query
sValue	String expression containing the value to query

Return Value

bFound	Boolean expression, True if the key value exist

Example

```
if KeyFound ("Customer", "LastName", "Smith") then
    InformUser "Key already exist!"
    Exit Sub
else
    SaveTheData
endif
```

Module: RUFDB.BAS Database functions

RUF Demos Programs

Two demo programs/projects are included with the RUF source code. RUFDEMO.MAK and RUFEDEMO.MAK both demonstrate the use of RUF reusable forms and a number of reusable functions. If you computer system does not use MSMAIL then you should use RUFDEMO.MAK which does not use the MSMAPI.VBX. Otherwise you can use RUFEDEMO.MAK that includes code demonstrating the use of RUF's e-mail functions. The projects require Visual Basic 3.0 Professional and you must supply the following .vbx files:CMDIALOG.VBX, THREED.VBX, MSMASKED.VBX, MSMAPI.VBX.

License Agreement

Reusable Functions for VB, Version 1.00

*** PLEASE READ THIS INFORMATION CAREFULLY ***
~~~~~~

TRIAL USE (SHAREWARE EVALUATION VERSION) LICENSE AND WARRANTY:
~~~~~~

Certain software from MIS Resources International is designed to produce source code modules or linkable objects. MIS Resources International does not warrant the suitability or merchantability of such code or objects when included in another product and assumes no liability for damages described below arising from the inclusion of such portions in other software.

If you include any RUF source code into an application for commercial sale you must register your copy of the RUF source code. Otherwise you may evaluate the RUF code during a 30 day trail period. You also may not incorporate parts of RUF code into your own library of functions.

The Shareware evaluation (trial use) version is provided AS IS.
MIS RESOURCES INTERNATIONAL MAKES NO WARRANTY OF ANY KIND, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE.