Sage 100 Customizer
Script Variables
Scripting-Plus

The following variables are available for use in scripts. These are passed into the script at run-time eliminating the need to declare these variables prior to use. Any variable that begins with an 'o' is a handle or reference to a Sage 100 object which gives the developer of the script access to a number of properties and methods that can be used to implement unique business rules.

Besides this reference guide that discusses the passed in object variables, the Object Reference section of the File Layouts help file is very useful for identifying object specific methods and properties including which arguments (parameters) are needed, and how many arguments to pass.

oBusObj

Object handle to the currently running business object.

- If the script is tied to the SalepersonNo field in AR_Customer, oBusObj will be a handle to the AR_Customer_bus object.
- If the script is tied to the QuantityShipped field in SO_SalesOrderDetail, oBusObj will be a handle to the SO_SalesOrderDetail_bus object.
- If the script is tied to the PreWrite() event of the CI_Item table, oBusObj will be a handle to the CI_Item_bus object

Useful Methods and Properties

retVal = oBusObj.GetValue(column$, val)	Obtains the value from the business object for the column requested (must be a column in the main table for the current business object, append a $ to the column for a string) and returns the value in val.
retVal = oBusObj.SetValue(column$, val)	Sets a new value into the business object for the column specified. retVal returns 1 if successful, -1 for a warning, or 0 for a failure. For 0 or -1 return values, you may check the oBusObj.LastErrorMsg property to see why the SetValue() call failed.
retVal = oBusObj.Write()	Commits changes to disk. retVal returns 1 if successful, -1 for a warning or 0 for a failure. For 0 or -1 return values, you may check the oBusObj.LastErrorMsg property to see why the Write() call failed.
retVal = oBusObj.Clear()	Takes a record out of edit state and discards any changes. For some objects where Write() doesn't release a locked record, Clear() will release it.
Set oChild = oBusObj.AsObject(oBusObj.GetChild Handle (Data Source as String))	Returns a handle to a service object for the requested data source. The data source is the name of the column (without the $) from the main table of the business object that is used to validate against another table. `Set oCust = oBusObj.AsObject( oBusObj.GetChildHandle("CustomerNo"))` will return the object

	handle to the AR_Customer_svc object. This is preferred over using the `oSession.AsObject(oSession.GetObject("AR_Customer_svc")` as the child handle object is already in memory. The GetObject() method of the Session object will create a new copy.
oBusObj.GetDataSources() as String	Returns a list of columns that validate against a service object. This is the same list you see under the Data Source dropdown in User Defined Field and Table Maintenance when creating a business object UDF. The columns are separated using the CHR(138) character (Hex 8A). `sDataSourceList = oBusObj.GetDataSources()`
oBusObj.ReadAdditional()	Reads all child data sources for current record
oBusObj.ReadAdditional(Data Source as String)	Reads a specified data source
oBusObj.EditState	EditState property values (0=no record in memory; 1=Existing record; 2=New record). Useful if you only want to run logic for a new record. Read Only cannot be set. (NOTE: This is also the return value of the SetKey() method.)
oBusObj.RecordChanged	Use this property to identify if the current record has changed (1) or is unchanged (0).
oBusObj.LastErrorMsg	This property will contain the reason for the last error that occurred. This property should be checked if any SetValue(), Write() or Delete() method calls return 0 to determine why those operations failed
oBusObj.Delete()	Use this to delete the current record from the business object. NOTE: As with the Write() method, this should not be called for the oBusObj handle, as the Delete() and Write() methods will be called during normal processing. If these methods were called from script, then the record would no longer be in an edit state and unexpected results would occur to the user. This is merely being documented to show how it can be used to update and delete rows from other business objects obtained with the GetObject() method of the oSession object.
retVal = oBusObj.GetValues(columns, data) New since v2013	Get multiple values from an object columns - A comma separated list of column names data - A Chr(138) separated list of data. `retVal = oBusObj.GetValues(“ItemCode,QuantityOrdered,ItemCodeDesc”, data)`
retVal = oBusObj.SetValues(columns, data) New since v2013	Set multiple values into object columns - A comma separated list of column names data - A Chr(138) separated list of data. Ex: `data="8953" & Chr(138) & "5" & Chr(138) & "Multi-Widget" & Chr(138)`

	`retVal = oBusObj.SetValues("ItemCode,QuantityOrdered,ItemCodeDesc", data)` **Note**: If the result of any of the values is a failure then retVal will be returned as a zero and a separated list of errors will be returned in oBusObj.LastErrorMsg

Useful Methods for Line Entry Detail Business Objects

retVal = oBusObj.AddLine()	Initialize a new line in the line entry object complete with any default values. Must be called before doing `SetValue()` calls to set other columns for the newly added line.

retVal = oBusObj.InsertLine(LineSeqNo)	To positionally insert a line (just like clicking the Insert Lines button on the Lines tab). Prior to this, do a GetValue() on the LineSeqNo column.
	
	Ex:
	
	`sLineSeqNo = ""`
	`retVal = oLines.GetValue("LineSeqNo$", sLineSeqNo)`
	`retVal = oLines.InsertLine(sLineSeqNo)`

retVal = oLines.Delete()	To delete a line or a header. Be careful with this one. If deleting a line, use oLines.Delete() if the starting point is the header object and use oBusObj.Delete() if the starting point is the lines object.

sEditKey = oLines.GetEditKey(LineKey)	Use this to get the EditKey value for the EditLine() method. Prior to this do a GetValue() on the LineKey column. Ex:
	
	`sLineKey = ""`
	`retVal = oLines.GetValue("LineKey$", sLineKey)`
	`sEditKey = oLines.GetEditKey(sLineKey)`

retVal = oLines.EditLine(sEditKey)	Use this to edit an existing line. Ex: `retVal = oLines.EditLine(sEditKey)` Now do a SetValue() on the columns that need changing the follow it up with a Write()

oSession

Object handle to the currently running session object.

Useful Methods and Properties

oSession.CompanyCode	Current company code
oSession.CompanyKey	Current company key from Sy_Company. Useful when using report object requiring key
oSession.CompanyName	Current company name
oSession.UserCode	Current user
oSession.AsObject(oSession.Security).IsMember("rolename")	Will return 1 if user belongs to specified security role or 0 if not a member. Useful for scripting based on security roles. (NOTE:

	IsMember is actually a method of the Security object which is a property of the Session object.)

SET oMyObj = oSession.AsObject (oSession.GetObject(objectName [,UDTableName]))	Returns a handle to the requested object into oMyObj using the security rights for the current user. UDTableName is only required if obtaining a business or service object for a User Defined Table. The AsObject method indicates that the return value is an object handle. (NOTE: If a user does not have sufficient security access for the requested object, then oMyObj will return as zero causing the SET to fail and crash the script. If it is possible the user does not have rights, the preferred technique is the following:
	`oMyObj = oSession.GetObject(objectName [,UDTTableName])`
	`if oMyObj <> 0 then`
	`     Set oMyObj = oSession.AsObject(oMyObj)`
	`end if`

oSession.DropObject(obj handle)	If in a line detail bus obj lot of re-use so may be better to not drop. Clean-up will occur anyway. DropObject() only runs in a button script or from external BOI.

oSession.Updating	Are we in the middle of an update?
	Value returned as numeric
	1 = Yes 0 = No
	Ex: A Post-Delete script runs in P/O ROG Entry. However this script should be excluded from P/O Daily Receipt Register Update.
	`If oSession.Updating = 0 Then`
	`   ‘ Run script code`
	`End If`

oSession.ModuleDate	Current module date (YYYYMMDD)
oSession.SystemDate	Current system date (YYYYMMDD)
oSession.PathRoot	Gives location of current installation Sage 100 directory. It is useful for relative path names to external files such as PDF documents.
retVal = oSession.FormatDate() *and* **retVal = oSession.GetFormattedDate()**	Needed to do date calculations on Sage 100 dates, using the VB Script DateAdd() function for example.
oSession.GetParameter(module, option_column, val)	Gets value from options table. Ex: `oSession.GetParameter(“A/R”, “Divisions$”, val)` returns “Y” if current company is set to use divisions.

Other Session Object Properties

oSession.UserName	User Logon field from User Maintenance oSesson.UserCode is the 3-digit User ID
oSession.CS (num)	1 = Adv / Premium 0 = Std
oSession.ModuleCode	Module code (e.g. A/P). For Library Master it is SYS
oSession.ModuleName	Module name (e.g. Accounts Payable)
oSession.CurrentPID	Server PID of current Sage 100 task (numeric)
oSession.CSHostIP	Sage 100 Adv/Prem App Server Port (string)
oSession.CSHostName	Sage 100 Adv/Prem Server name (string)
oSession.WorkstationName	Workstation Computer Name (string). If running from Terminal Server / Citrix the Computer Name of remote PC is returned

oSession.StartProgram	Returns start program name the session belongs to (string)
	Used to condition scripts
	Ex: When a Post-Write script runs in S/O Entry it performs certain tasks that should not be performed where orders are auto-created such as RMA Generation, Auto Generate Sales Order, EDI imports, structured web imports (e.g. In-Synch, Website Pipeline), or VI imports.
	
	Note: StartProgram should not be used to control whether it is safe to perform UI logic such as disable/hide controls, message dialogs, Etc. because in most BOI type applications, the StartProgram is set to the same value as when running from the Sage 100 ERP desktop. See the section under oUIObj for the recommended method for UI detection.
	
	`If oSession.StartProgram = "SO_SALESORDER_UI" Then`
	`   ‘ Run script code`
	`End If`

oScript

Object handle to the script helper object.

There is a separate script object for each business object for which scripts have been tied to events using User Defined Scripts. For example, SO_InvoiceDetail_bus has a script object associated with it, and SO_Invoice_bus (header object) has its own script object. Also, scripts run from the User Interface using a Customizer BT_Link script button have a separate script object.

Useful Methods

retVal = oScript.SetStorageVar (id_desc as String, val)	Use this method to store any number of values that need to be accessible across function calls. Meaning they can be set in one procedure and obtained in another procedure for processing. Very useful for performance purposes when setting default values programmatically. (NOTE: object handles cannot be stored as objects, only numerics)
retVal = oSession.SetStorageVar (id_desc as String, val)	Use this variation when you have “layers”. For example if you have a line event script running but a Lot/Serial Distribution window appears or a developer window

	appears, use the oSession version of SetStorageVar.
retVal = oScript.GetStorageVar (id_desc as String, val)	Complement to SetStorageVar, used to get a previously stored value.
retVal = oSession.GetStorageVar (id_desc as String, val)	Use this variation when you have "layers". For example if you have a line event script running but a Lot/Serial Distribution window appears or a developer window appears, use the oSession version of GetStorageVar.
retVal = oScript.SetError(errstring as String)	Used to fail any of the "Pre" procedures – Pre-Validation, Pre-Write, and Pre-Delete. Set the reason you are failing the procedure into the errString argument. **Note:** After setting the error, you should use an Exit Sub to halt any further processing of the script. The business framework base classes will then evaluate this error message and display the appropriate error message to the user or, in the case of VI, will write to the error log the reason for the failure.
retVal = oScript.SetWarning (warnstring as String)	Same as SetError only the method will not fail and processing will continue. The warning message box will still be displayed to the user. (NOTE: because further processing will occur, it is possible that the warning message will be overridden by standard Sage 100 logic.)
retVal = oScript.ActivateProcedure (proc_name as String) *and* **retVal = oScript.DeactivateProcedure (proc_name as String)**	These are used to give the script author the ability to avoid recursive calls. For example, if the PostValidateQuantityShipped(col, val) procedure script was to invoke the oBusObj.SetValue("QuantityShipped", val), this would in turn cause the PostValidateQuantityShipped(col, val) procedure to be called again. To avoid this, call retVal = oScript.DeactivateProcedure("PostValidateQuantityShipped") prior to the SetValue() call, then use the retVal=oScript.ActivateProcedure("PostValidateQuantityShipped") to reactivate this procedure. **Note:** Use retVal = oScript.DeactivateProcedure("*ALL*") to deactivate/activate all procedures for the current business object. **Note:** Use retVal = oScript.Deactivate("*") to deactivate/activate the current procedure.
retVal = oScript.InvokeButton (btn_name as String)	Used to invoke a button. From a business object script this will only fire after the script is complete AND if there is a UI object present. This is to prevent any UI during a VI job or external use of the business object. The button name to be invoked can be determined by looking at the control names when editing a form in Customizer Selection. Tab folders are also considered buttons and can be invoked as well. The name of the folder button is preceded by fldr.<panel_name> `retVal = oScript.InvokeButton("fldr.pMain")` will simulate the user clicking on the Main tab folder. From a UI script (i.e. BT_Link), buttons can be fired immediately by using the `retVal = oUIObj.HandleScriptUI()` immediately after calling the `retVal =`

	`oScript.InvokeButton("BT_Accept")` method.
oScript.LinesAdded	Use this property to set the number of lines added using your script. Make sure you are checking the return value on the Write() method of the lines business object to ensure the line was actually added when setting this property.
retVal = oScript.LoadGrid (grid_control as String)	This is only required if the grid you are attempting to load is not "GD_Lines" (which is the default grid in all Sage 100 data entry screens). The UI object will automatically attempt to load GD_Lines if the oScript.LinesAdded property is not zero upon exiting the script.
retVal = oScript.SetUIControl (control as String, action as String)	Used to perform a specific action on a given control. Control is name of control - for example, "BT_Link_1" - and action can be one of the following: ▪ **ENABLE:** Enable control ▪ **DISABLE:** Disable control ▪ **SHOW:** Show control ▪ **HIDE:** Hide control At run-time the system detects whether or not a Sage 100 screen is in use. If not (such as when using VI to import customers), then these calls are ignored. Since the scripter may not know if the user if on the specific tab folder for which the control is being hidden or shown, a list of controls is maintained so that if the user switches to a tab that has a control that should be hidden it will be hidden. Caveat: In some cases there may be subsequent logic either from standard Sage 100 code or from Master Developer logic that will run after the script has run, causing a field to be re-shown after the script was run to hide it. This can happen on standard Sage 100 fields. In these cases the recommended work-around is to hide the fields in Customizer and place them on a new Link dialog. Then the script can hide and show the BT_Link_1 button. **Note:** Only works on controls, will NOT work for grid cells.
retVal = oScript.DebugPrint(text)	Method to output variables and text to aid in debugging scripts. This requires use of the Providex trace window. This trace window can be enabled by adding 'Debug=1' in the SOTA.ini file in the Launcher directory under the [config] section. Once enabled right click on the Sage 100 title bar of the current task (e.g. AR Customer Maintenance) and select Debugging Environment..Program..Trace Window. In the trace window choose Options..Suppress Program trace. When the scripts are running, any text from a DebugPrint() call will be displayed in the trace window.
currentProc = oScript.GetCurrentProcedure() New in v2015	Returns the current procedure that the script is running for. Useful when pinning the same script to multiple events.

oHeaderObj

Object handle to the header object for the current detail object.

Only available on detail business objects, for example, SO_SalesOrderDetail_bus scripts have oHeaderObj as a handle to SO_SalesOrder_bus. This can be useful in setting default values on lines columns based on a header value. Also useful for setting a header column based on some script in a detail line, for example, to set UDF_DropShipNeeded$ on the header in the PostValidateDropShip procedure of the detail line.

column, value

These two variables are passed in for the two column-level Pre-Validate and Post-Validate procedures. **column** contains the name of the column for which this procedure was called. It is stripped of the dollar-sign if it is a string (e.g. SalesPersonNo). This can then be used to get a child handle for any column that validates against another Sage 100 table or UDT. **value** contains the value that was set into the business object.

oUIObj

Object handle to the currently running UI object.

By default this is only available in the context of UI scripts and button scripts. However it may be available in the context of table event scripts ***IF*** the business object is being run from the Sage 100 Erp UI task. It is the responsibility of the script programmer to implement proper checking to ensure accurate detection of when the UI object is available.

The following debug script, used for QA purposes, illustrates the recommended method for UI detection:

```
' Recommended UI detection - if no UI then do not use any type of MsgBox or other UI.
' This is important because if your script pops a MsgBox during an automated process
' such as an import, or webservices running as a background process, the service will hang.

If (IsObject(oUIObj)) Then
     ' This is either a UI Event Script or a Button link script
    ' – because oUIObj is directly available.
     MASUI = True
     screenName = oUIObj.GetScreenName()
     panelName = oUIObj.GetPanelName()
     folderName = oUIObj.GetFolderName()
Else
     ' This is a Business Event script. Must test if UI is present/available
     MASUI = CBool(oScript.UIObj)
     If (MASUI) Then
             ' Need to get my own handle to access oUIObj functionality such as InvokeChange(), InvokeLookup(), Etc.
             Set myUIObj = oSession.AsObject(oScript.UIObj)
             screenName = myUIObj.GetScreenName()
             panelName = myUIObj.GetPanelName()
             folderName = myUIObj.GetFolderName()
             Set myUIObj = Nothing
     End If
End If

currentProc = oScript.GetCurrentProcedure() ' Available in both UI and Business Events - no need to check for UI.

If MASUI Then
     ' Ok to Message box
     uiContext = "Current Proc: " & currentProc & vbCRLF & "Screen Name: " & screenName & vbCRLF & \
                 "Panel Name: " & panelName & vbCRLF & "Folder Name: " & folderName
     ' Always use Sage 100 MessageBox instead of VbScript MsgBox() to avoid msg appearing on
     ' the server where no one can click on it
     oSession.AsObject(oSession.UI).MessageBox "", uiContext
Else
     ' Not Ok to Message box
     tableName = oBusObj.GetTable("main")
     busContext = "Current Proc: " & currentProc & " - Table Name: " & tableName
    ' But Ok to print to Trace Window.
     oScript.DebugPrint busContext
    ' And A-Ok to write to the activity log.
    oSession.WriteLog "A", "Yay! Scripting can write to the activity log! " & busContext
End If
```

Note: oBusObj, oSession, oScript, oLines object handles are also available to button scripts as long as the script is set to *Execute Script on the Server*.

Useful UI Methods

retVal = oUIObj.HandleScriptUI() Use this to immediately fire off any UI. Always use after InvokeButton()	Related events that were requested using the oScript objects, for example, InvokeButton(), SetUIControl(), or LoadGrid().
retVal = oUIObj.GetValue(ctlName , val)	Used to obtain the value of a control on the screen (the control ID). You can see the names of the controls that are available from within Customizer Selection when editing a panel. `retVal = oUIObj.GetValue("ML_SourceJournal$", val)`

retVal = oUIObj.InvokeChange(ctlName, val [,gridName])
 	
New since version 4.50
	Used to change the value of a control on the screen or a column in a grid. This method may be useful for certain situations where there is unique logic associated with changing the value of a control on the screen or in a grid, which is not invoked via oBusObj.SetValue(). E.g. Changing the QuantityOrdered in a sales order line does not update the “Total Amount” displayed on the lower-right corner of the Lines tab.
	
	Arguments:
	
	ctlName: Required, String, the name of the multiline or grid column being changed, (no “$”).
	
	val: Required, String or Numeric: The new value to change the control to.
	
	gridName: Required when changing grid columns only. Example “GD_Lines”.
	
	Example: Changing the quantity ordered on a line.
	
	`retVal = oUIObj.InvokeChange("QuantityOrdered", numVal, "GD_Lines")`
	
	Example: Changing the customer number on a sales order header:
	`retVal = oUIObj.InvokeChange("ML_Customer", strVal)`

retVal = oUIObj.SetFolderState(tab folder, action) New since v2013	Allows you to disable or enable tab folders. `retVal = oUIObj.SetFolderState("pAddress,pLines,pTotals", "ENABLE")`
retVal = oUIObj.InvokeLookup(lookupCode, value, [startValue]) New since v2013	Invoke a Sage 100 lookup from a button script lookupCode = A valid Sage 100 lookup code value = the value selected by user from the lookup startValue = Use if a lookup requires a starting key value. Note: if dealing with a multi-part key, general rule is to null pad all the key segments except the last.

	`retVal = oUIObj.InvokeLookup("CI_ItemAll", item)` Note: Only available for custom button scripts. If no value was selected or an invalid lookup code, then the return value will equal zero.

returnValue = oUI.ProgressBar(option as string, title as string, msg as string, pct as number, extra options as string)
 	
<u>Option has 3 choices:</u>
init - sets up the dialogue to display progress
update - increments the progress bar
close - terminates the progress bar
	Allows you to show you a progress meter. Useful if you have a long running script. Note in a button script usually you have to set the UI object handle separately:
	`If oSession.UI<>0 Then`
	`  Set oUI = oSession.AsObject(oSession.UI)`
	`End If`
	
	`returnValue = oUI.ProgressBar("init", "Updating AR Invc Hist Hdr...", "Update SP Comm ...", 0,"")`
	`' Follow it up with` *`update`* `and` *`close`* `statements`

oUIObj.GetControlProperty(ctlName, propertyName, value)
New in v2015
	Retrieve the value of a NOMADs control property. Available properties (depends on the control type) include: BackColour, Enabled, TextColour, Value, Visible.
	Example: Get number of rows in a grid
	`rows = ""`
	`oUIObj.GetControlProperty("GD_LINES", "RowsHigh", rows)`

oUIObj.SetControlProperty(ctlName, propertyName, value)
New in v2015
	Sets a property value to a NOMADs control property. Available properties and valid values depend on the control type.
	Example: Set the background color of the 2nd row in the grid.
	`oUIObj.SetControlProperty("GD_LINES","Row","2")`
	`oUIObj.SetControlProperty("GD_LINES","Column", "0")`
	`oUIObj.SetControlProperty("GD_LINES","Backcolor","RED")`

oUIObj.GetScreenName()
oUIObj.GetPanelName()
oUIObj.GetFolderName()
New in v2015
	Returns the current screen (Library) name, the current panel or dialog name, and the current folder tab.
	`screenName = oUIObj.GetScreenName()`
	`panelName = oUIObj.GetPanelName()`
	`folderName = oUIObj.GetFolderName()`

oUIObj.DropBoxLoad(dropBoxName, value, delimiter) New in v2015	Load or reload a drop box with a delimited list of values. This can be used with both factory and UDF drop boxes. **Arguments**: **dropBoxName**: Required, string. Name of the drop box being loaded. **value**: Required, string. Delimited list of values. **delimiter**: Optional, string. Delimiter used in the list of values Example: Load a drop box UDF named DROPBOX with the values One, Two, and Three. `oUIObj.DropBoxLoad "UDF_DROPBOX", "One/Two/Three/"` Note: Trailing delimiter required. Setting `%NOMAD_Suppress_ListErr$` (required for UDF drop boxes) Because UDF drop boxes are validated against the list of values (or no value) entered when creating the UDF, so it is necessary to suppress the error generated when dynamically loading a different set of values. This only

	needs to be done once per session, typically in the PostLoadDMain() event. This is an example of setting this value for a UDF list box named LISTBOX and a drop box named DROPBOX2: `oUIObj.SetVar "%NOMAD_Suppress_ListErr$", "/UDF_LISTBOX/UDF_DROPBOX2/"`

oUIObj.ListBoxLoad(listBoxName, value, delimiter)
New in v2015
	Load or reload a list box with a delimited list of values.
	Arguments:
	listBoxName: Required, string. Name of the list box being loaded.
	value: Required, string. Delimited list of values.
	delimiter: Optional, string. Delimiter used in the list of values
	Example: Load a list box UDF named LISTBOX with the values One, Two, and Three.
	`oUIObj.ListBoxLoad "UDF_LISTBOX", "One/Two/Three/"`
	Note: Trailing delimiter required.
	Setting `%NOMAD_Suppress_ListErr$` (required for UDF list boxes) Because UDF list boxes are validated against the list of values (or no value) entered when creating the UDF, so it is necessary to suppress the error generated when dynamically loading a different set of values. This only needs to be done once per session, typically in the PostLoadDMain() event. This is an example of setting this value for a UDF list box named LISTBOX and a drop box named DROPBOX2:
	`oUIObj.SetVar "%NOMAD_Suppress_ListErr$", "/UDF_LISTBOX/UDF_DROPBOX2/"`

Some Common Methods (any business object)

.IsMember("rolename")	Ex: Check if current user is member of SalesMgr role (as defined in Role Maint)
	`sRoleName = "SalesMgr"`
	`If oSession.AsObject(oSession.Security).IsMember(sURL) > 0 Then`
	`   ‘ …. Run script code`
	`End If`

.SetToReadOnly(text)	Will produce an "Unable to Edit" message followed by any user text and prevents any access to that bus obj.

.MessageBox(btn as string, msg as string)	Produces message box. Must use this variation for Adv/Prem and recommended for Std version too:
	`sMsg = "This is the MessageBox method instead of MsgBox VB fcn"`
	`retMsg = oSession.AsObject(oSession.UI).MessageBox("", sMsg)`

.MoveFirst()	Move to the first record in the bus object
.MoveLast()	Move to the last record in the bus object
.MovePrevious()	Move to the previous record in the bus object
.MoveNext()	Move to the next record in the bus object
.SetBrowseFilter(filter)	Pre-filters the next MoveNext() or MovePrevious() e.g. In SO_SalesHistory you have the 1st two keys but only a partial

	value of the 3rd key but doing MoveFirst() then MoveNext() is time consuming. `retVal= oSalesHistory.SetBrowseFilter(sDiv & sCust & "NGK")` Now when doing oSalesHistory.MoveNext() or oSalesHistory.MoveNext() it will not go to next sequential record but filtered based on the division in sDiv, customer number in sCust, & item code starting with "NGK".
.SetBrowseIndex(NewBrowseIndex, BrowseIndex)	Allows you to assign a new index for the MoveNext() or MovePrevious()

.Find(key)	Allows you to do a keyed lookup to find a value. SetKey() also does but use that when you need edit the record or access the lines portion of header / detail object. If you just need to grab a header value and not change it, use Find.
	
	e.g. `retVal = oGLAccount.Find(tmpAcctKey)`
	
	This finds the primary key in GL_Account.m4t that matches tmpAcctKey.
	
	Note: If the key contains multiple key segments there are 2 ways to deal with this:
	
	1) Concatenate the key segment values but general rule is all but the last key segment needs to be null padded. Ex:
	`PaddedKey = sItemCode & String(30-Len(sItemCode),Chr(0)) & sWhse`
	`retFind = oItemWhse.Find(PaddedKey)`
	
	2) Use the _bus object, then SetKeyValue() on each key segment, then instead of SetKey() to put the record in an edit state (which is okay if you want to edit the record or create a new one), use Find() without an argument.

.SetKeyValue(key column as string, key value as string)	Primarily used prior to the SetKey() command with no arguments passed in. Use this to set each key segment when you have multi-part keys.
	
	Ex 1 : Assume a script runs in A/R Customer Contact Maintenance
	
	`retVal = oBusObj.SetKeyValue("ARDivisionNo$", sDiv)`
	`retVal = oBusObj.SetKeyValue("CustomerNo$", sCust)`
	`retVal = oBusObj.SetKeyValue("ContactCode$", sContactCode)`
	`'Now the do the SetKey() with no arguments:`
	`retVal = oBusObj.SetKey() 'Now the row has been put into an Edit State`
	
	Use SetKeyValue() also on a business object when you need to do a Find()with multi-part keys:
	
	Ex 2: Assume you need to Find() an item code and warehouse code and have done a GetObject() to create an object handle to I/M Item Warehouse Code as oItemWarehouse:
	
	`sItemCode = "6655" : sWhse = "000" : QOH = 0`
	`retVal = oItemWarehouse.SetKeyValue("ItemCode$", sItemCode)`
	`retVal = oItemWarehouse.SetKeyValue("WarehouseCode$", sWhse)`
	`retVal = oItemWarehouse.Find()`
	`retVal = oItemWarehouse.GetValue("QuantityOnHand", QOH)`

.SetKey(key value as string) or .SetKey() with no arguments	Use this to put the record into an Edit State: 2 = New record 1 = Existing record 0 = No record in memory

	Ex 1:
	`sItemCode = "6655"`
	`retVal = oBusObj.SetKey(sItemCode)`
	`Select Case retVal`
	`Case 2`
	` ‘Allows you to creates a new item`
	`Case 1`
	` ‘Update a field on existing item via SetValue() and Write()`
	`End Select`
	
	Ex 2: For multi-part keys use SetKeyValue() from above

.GetKeyColumns() as String	Allows you to find the key columns on a table. You can also look at File Resources for this purpose:
	
	`sKeyCols = oBusObj.GetKeyColumns()`
	`retMsg = oSession.AsObject(oSession.UI).MessageBox("", sKeyCols)`

.GetColumns(Data Source) as String	Returns a CHR(138) separated list of all columns from the main data source or an alternate data source for an object
	
	`sCols = oBusObj.GetColumns("MAIN")`
	`retMsg = oSession.AsObject(oSession.UI).MessageBox("", sCols)`
	
	If needed, use the VB Replace functions to substitute the CHR(138) characters (Hex 8A) with something more exportable such as commas:
	
	`sCols2 = Replace(sCols, CHR(138), ",")`
	`retMsg = oSession.AsObject(oSession.UI).MessageBox("", sCol2)`

.GetRecord(<record data as String>, pvx IOList>)	This is not useful in scripting. Use GetColumns() and/or GetValues() instead

.BOF and .EOF properties	Indicates whether or not the record pointer is at the begging or end of the file, respectively. Very commonly used in looping structures.
	
	Ex: When looping through a Lines grid typical code is:
	`Set oLines  = oSession.AsObject(oBusObj.Lines)`
	`retVal = oLines.MoveFirst()`
	`Do Until cBool(oLines.EOF) ‘Keep looping until record pointer is on last line`
	`..`
	`..`
	`  retVal = oLines.MoveNext()`
	`Loop`

ScriptTimeout property	The default ScriptTimeout value is 10 seconds. It indicates how long to wait before sending timeout signal to the Windows Scripting Host. Use this to extend processing for scripts that may take longer to execute. Note this is a global setting for all users. To make this change edit the ..\mas90\Launcher\Sota.ini file on the server and add an entry to the [Config] section: [Config] Serial=… Users=… … … ScriptTimeout = 30000 ‘In milliseconds - changes timeout to 30 seconds
.SetIndex(Index Name as string)	If you know the primary key or an alternate key, this is much faster search than MoveNext() which is a sequential positioning on the next record (unless a browse filter was previously set through SetBrowseFilter). Ex: Since GL_Account table has multiple keys/indexes, you can take advantage of using a different search index for speed.

	If you want to search by fully formatted account (FFA) instead of the primary key AccountKey:
	`retVal = oGLAccount.SetIndex("KACCOUNT")`
	`retval = oGLAccount.Find(tmpAcctNo)`
	
	For a list of indexes for a table use either DFDM or File Layouts.

IsObject(VB object variable)
 	
Set (VB variable as an object variable)
	IsObject() is VB command to check if variable is also an object handle
	Returns True or False (implicitly or explicitly)
	
	Ex 1: Check if we have an object handle PO_PurchaseOrder_bus already. If not then get an instance of the object:
	`If Not(IsObject(oPO)) Then`
	` Set oPO`
	`=oSession.AsObject(oSession.GetObject("PO_PurchaseOrder_bus"))`
	`End If`
	
	Ex 1a - Alternate:
	`If IsObject(oPO) = False Then`
	` Set oPO`
	`=oSession.AsObject(oSession.GetObject("PO_PurchaseOrder_bus"))`
	`End If`
	
	If the Sage 100 session stays open (user does not close down screen) in most cases you can re-use existing object.
	
	To get a new copy of an object use the GetObject() command but first check to see if a data source exists (same Data Source you see in UDF Maintenance in Custom Office). If it does use that GetChildHandle() command instead because you are taking advantage of the same object currently in memory that is pre-linked to another table.

.SetCopyKeyValue(<keyColumn As String>, <keyValue As String>)	Set the individual key segment to prepare for the CopyFrom()
	Ex:
	`retVal = oBusObj.SetCopyKeyValue("ARDivisionNo$", sDiv)`
	`retVal = oBusObj.SetCopyKeyValue("CustomerNo$", sCust)`
	`retVal = oBusObj.SetCopyKeyValue("ContactCode$", sContactCode)`

.CopyFrom(key) or .CopyFrom()	Copies all the non-key fields into an object. If SetCopyKeyValue() was used to set the individual key segments then no argument is needed Example continued from SetCopyKeyValue() above: `retVal  = oBusObj.CopyFrom()`