OPENSCRIPT & ACTIONS EDITOR
PROGRAMMING REFERENCE GUIDE

for

TOOLBOOK INSTRUCTOR 2004

Written by Denny Dedmore
Third Edition • August 2005
www.toolboolhelp.com

PREFACE

As I was learning how to use ToolBook 4.0, having a programmers reference manual that I could read through while sitting on the couch or on the bus was a huge convenience. However, although ToolBook used to ship with an OpenScript Reference Manual, SumTotal Systems decided to no longer offer that manual starting with ToolBook 5.

Many in the ToolBook community, including myself, have since desired that it once again become available as well as having it incorporate the new features ToolBook has introduced over the years. This book is intended to do just that.

This guide provides reference to over 1,000 ToolBook terms, commands and functions, and provides quick access to them via the thoroughly cross-referenced Index at the back of this book.

In an attempt to keep this manual to a reasonable size, you will find that not every ToolBook term is covered. For those terms you don't find in this manual, please consult your online Help.

Although this book is intended for ToolBook Instructor 2004 users, much of the information is valid for any version of ToolBook, making it a perfect reference guide for any ToolBook user.

Please note that this manual is not intended to provide instruction on the use of ToolBook. It is simply a programming reference guide for those who are interested in programming using OpenScript or the Actions Editor.

For those of you interested in Actions Editor programming, look up *Actions Editor* in the INDEX.

I welcome any feedback you may have about this manual. I do plan to update it periodically so your feedback will help to produce a better reference manual. Send all feedback to `toolbookhelp@yahoo.com`. You can also visit my web site at `www.toolbookhelp.com`.

ABOUT THE AUTHOR

Many in the ToolBook community already know me from my involvement on the ToolBook ListServ, the ToolBook Newsgroups and the Platte Canyon ToolBook user conferences. However for the record, here is my history of ToolBook involvement.

- My name is Denny Dedmore and I have worked for SumTotal Systems for the last 9 years.
- I joined the ToolBook technical support team back in 1996 when the company was Asymetrix, shortly after ToolBook 4 shipped, not knowing the first thing about ToolBook. As you can well imagine, I quickly learned.
- Over the next three years I worked with many of you via email and phone to resolve your ToolBook issues. I even created a Web Site and maintained it for 2 years using ToolBook 5 to provide users with detailed information in working out their ToolBook 5 and 6 HTML export problems.
- I created a 3rd party ToolBook plug-in called FTS Pro which provides Full Text Search abilities in ToolBook. This utility has been, and still is, available from Platte Canyon for several years now.
- My 4th year with the company was in the role of Technical Support Manager.
- In the 5th and 6th years with SumTotal Systems were in the capacity of an OpenScript developer/engineer working on ToolBook 8.0, 8.1 and 8.5.
- I now work in the Technical Support department again as the Senior Support Engineer for the ToolBook product line.
- I created www.toolbookhelp.com in June of 2002 and maintain it still.

The nature of ToolBook has changed quite a bit over the years, from being an Application Development software package to a Web Training software package. Although much has been done starting with version 5 to make ToolBook web enabled, I have to admit that my primary interest in ToolBook lies in its power as an Application Development software package - where OpenScript can be utilized to create virtually any application you may need to create.

TOOLBOOK FUNCTIONS WHICH NO LONGER REQUIRE TO BE LINKED

As of ToolBook 8.5, many of the functions found in the WIN, DOS, DLG, REG and TBFILE32 DLLs have been converted to internal functions. This means that you no longer need to LINK these functions before using them.

chooseColorDlg
chooseDirectoryDlg32
chooseFontDlg
clientFromPage
clientFromScreen
colorPaletteDlg
copyFile32
createDirectory32
displayAspectX
displayAspectXY
displayAspectY
displayBitsPerPixel
displayColorPlanes
displayFonts
displayLogPixelsX
displayLogPixelsY
fileExists32
fileToPrinter
getCDDriveList32
getCurrentDirectory32
getCurrentDrive32
getCustomColors
getDirectoryOnlyList32
getDOSEnvironmentString32
getDriveKind32
getDriveList32
getEllipsisByCharCount32
getEllipsisByFont32
getFileAttributes32
getFileDate32
getFileDateEx32
getFileList32
getFileListDlg
getFileListDlgFilterIndex
getFileOnlyList32
getFileSize32
getFileVersion32
getFreeDiskSpace32
getIniVar
getLongFileName32
getModuleList
getModulePath
getOpenFileDlgFilterIndex32
getSaveAsDlgFilterIndex32
getShortFileName32
getVolumeName32
getWinIniVar
HLStoRGB
horizontalDisplayRes
horizontalDisplaySize
isCDDrive32
isDOSDrive32
itemOffset
listToTextline
modalPopText
moveFile32
openDlgLFN
openFileDlg32
pageFromClient
pageFromScreen
popMenu
popText
popTextGetBounds
printerFonts
registryGetKey
registryGetValueSize
registryGetValueType
registryKeyList
registryRemoveKey
registryRemoveValue
registrySetKey
registryValueList
removeDirectory32
removeFile32
RGBtoHLS
saveAsDlg32
screenFromClient
screenFromPage
sendKeys
setCurrentDirectory32
setCurrentDrive32
setCustomColors
setFileAttributes32
setFileDate32
setIniVar
setSystemCursor32
setSystemDate32
setSystemTime32
setWinIniVar
sortList
sortTextlines
textlineContains
textlineOffset
textlineToList
textToPrinter
validateForDOS32
verticalDisplayRes
verticalDisplaySize
xPixelsFromUnits
xUnitsFromPixels
yieldApp
yPixelsFromUnits
yUnitsFromPixels

TOOLBOOK FUNCTIONS WHICH STILL REQUIRE TO BE LINKED BEFORE USING THEM

As of ToolBook 8.5, many of the functions found in the WIN, DOS, DLG, REG and TBFILE32 DLLs have been converted to internal functions. However, some have **not** been converted.

These functions have not been converted to internal functions, so you will still be required to link to them as you have in the past – OR – you can use the suggested alternate function. The suggested replacement function will be an internal function which will not require any linking.

Those functions that have not been converted to internal functions will *no longer work* as of version 2006 of ToolBook since linking to 16-bit DLLs will not be permitted in version 2006 since ToolBook will then be a 32 bit product. As such it is a good idea to avoid using these functions and instead start using the suggested replacement functions.

FUNCTION NAME	SUGGESTED REPLACEMENT
chooseDirectoryDlg	chooseDirectoryDlg32
chooseDirectoryDlgLFN	chooseDirectoryDlg32
copyFile	copyFile32
createDirectory	createDirectory32
fileExists	fileExists32
freeMemBlock	freeMemBlock32
getCDDriveList	getCDDriveList32
getCurrentDirectory	getCurrentDirectory32
getCurrentDirectoryLFN	getCurrentDirectory32
getCurrentDrive	getCurrentDrive32
getDirectoryOnlyList	getDirectoryOnlyList32
getDirectoryOnlyListLFN	getDirectoryOnlyList32
getDosEnvironmentString	getDosEnvironmentString32
getDriveKind	getDriveKind32
getDriveList	getDriveList32
getFileAttributes	getFileAttributes32
getFileDate	getFileDate32
getFileList	getFileList32
getFileListLFN	getFileList32
getFileOnlyList	getFileOnlyList32
getFileOnlyListLFN	getFileOnlyList32
getFileSize	getFileSize32
getFileVersion	getFileVersion32

FUNCTION NAME	SUGGESTED REPLACEMENT
getFreeDiskSpace	getFreeDiskSpace32
getMemBlock	getMemBlock32
getOpenFileDlgFilterIndex	getOpenFileDlgFilterIndex32
getSaveAsDlgFilterIndex	getSaveAsDlgFilterIndex32
getVolumeName	getVolumeName32
isCDDrive	isCDDrive32
moveFile	moveFile32
openDlg	openFileDlg32
openFileDlg	openFileDlg32
openFileDlgLFN	openFileDlg32
registryGetBinaryKey	registryGetBinaryKey32
registrySetBinaryKey	registrySetBinaryKey32
removeDirectory	removeDirectory32
removeFile	removeFile32
saveAsDlg	saveAsDlg32
saveAsDlgLFN	saveAsDlg32
setCurrentDirectory	setCurrentDirectory32
setCurrentDrive	setCurrentDrive32
setFileAttributes	setFileAttributes32
setFileDate	setFileDate32
setSystemDate	setSystemDate32
setSystemTime	setSystemTime32

BACKGROUND

OPENSCRIPT

MESSAGES

make
destroy

PROPERTIES

backdrop
backdropStyle
fillColor
idNumber
imageInvalid
name
notifyObjects
object
objectCount
objects
pageCount
pages
parent
pattern
percentFreeSpace
rgbFill
rgbStroke
script
sharedScript
size
storedImages
storeImage
strokeColor
uniqueName
useDialogColor
userProperties
useWindowsColors

ACTIONS EDITOR

EVENTS

click
double-click
load background
load page
property change
reset
right-click
trigger
unload background
unload page
user event

PROPERTIES

height
name
rgbFill
width

NOTES

- The size of a background controls the size of the pages used by that background.
- Pages don't have a size.
- A size of 0,0 tells the background to use the size of the book.
- The activated property of fields on a background is always true.
- Although you would think enterBackground and leaveBackground would be sent to the background, it is actually sent to the current Page.

BOOK

OPENSCRIPT

MESSAGES

enterBook
leaveBook

PROPERTIES

activeCacheFile
backgroundCount
backgrounds
bookURL
bounds
buildCacheFile
cacheFileType
caption
CDMediaPath
controlStyle
customColors
footer
HDMediaPath
isChanged
keepMenuBar
majorVersionNumber
minorVersionNumber
name
header
hotwordColor
hotwordStyle
icon
object
pageCount
pages
palette
saveOnClose
script
sharedScript
size
solidColorsEnabled
uniqueName
userProperties
windows
wordWrapUnits

ACTIONS EDITOR

EVENTS

load book
property change
reset
trigger
unload book
user event

PROPERTIES

description
name
studentName
title

NOTES

- In OpenScript, you can get the full path of your book by requesting the “name of this book”.
- When a book is opened it will show page 1 by default. If you want to change which page is initially shown, set the defaultPage of the MainWindow.
- The size of a book can be compacted if you perform a Save As to a different file name.
- Using CTL-SHIFT-S to save your book will save your book and compact it too. <undocumented>

BUTTON

OPENSCRIPT

MESSAGES

make
destroy
enterButton
leaveButton
moved
sized
buttonClick
buttonDown
buttonUp
buttonDoubleClick
buttonStillDown
rightButtonDown
rightButtonUp
rightButtonDoubleClick
mouseEnter
mouseLeave
keyChar
keyDown
keyUp

PROPERTIES

borderStyle
bounds
caption
captionPosition
checked
checkedGraphic
defaultAllowDrag
defaultAllowDrop
disabledGraphic
dragImage
drawDirect
enabled
excludeTab
fillColor
fontFace
fontSize
fontStyle
highlight
idNumber
invert
invertGraphic
layer
name
noDropImage
normalGraphic
notifyAfterMessages
notifyBeforeMessages
object
parent
position
rgbFill
rgbStroke
script
sharedScript
size
stretchGraphic
strokeColor
textOverflow
textUnderflow
transparent
uniqueName
userProperties
useWindowsColors
vertices
visible

ACTIONS EDITOR

EVENTS

click
double-click
key down
key up
key press
load page
mouse off
mouse over
property change
reset
right-click
trigger
unload page
user event

PROPERTIES

caption
checked
enabled
height
name
rgbFill
selectedItemText
itemSelected
itemText
left
top
transparent
visible
width

NOTES

- You can add an image to a button by setting the normalGraphic property.
- To add a mnemonic access character to a button caption, precede the character with a & character (example: caption of button “foo” = “Exit &Now”).
- All the different button styles in ToolBook are just variations of the Button object type. To change from one button style to another, simply change the borderStyle of the button.

COMBOBOX

OPENSCRIPT

MESSAGES

make
destroy
enterComboBox
leaveComboBox
enterDropDown
leaveDropDown
selectChange
moved
sized
buttonClick
buttonDown
buttonUp
buttonDoubleClick
buttonStillDown
rightButtonDown
rightButtonUp
rightButtonDoubleClick
mouseEnter
mouseLeave
keyChar
keyDown
keyUp

PROPERTIES

borderStyle
bounds
defaultAllowDrag
defaultAllowDrop
dragImage
drawDirect
dropDownItems
editable
enabled
fillColor
fontFace
fontSize
fontStyle
idNumber
layer
lineCount
name
noDropImage
notifyAfterMessages
notifyBeforeMessages
object
parent
position
rgbFill
rgbStroke
script
scrollable
selectedItem
sharedScript
size
sortItems
strokeColor
text
textOverflow
textUnderflow
transparent
uniqueName
userProperties
useWindowsColors
vertices
visible

ACTIONS EDITOR

EVENTS

key down
key press
key up
load page
mouse off
mouse over
property change
reset
right-click
select
trigger
unload page
user event

PROPERTIES

height
itemSelected
itemText
left
name
rgbFill
selectedItemText
top
visible
width

NOTES

- You can alphabetize the dropdown items in your combobox by setting the sortItems property.
- You can control whether a user can type into a combobox by setting the editable property.
- The number of dropdown items visible in the dropdown list is controlled by the lineCount property.
- In DHTML you will be unable to dynamically set the items in the dropdown list to be more or less than the number of items which were present when exported.

DRAW OBJECT

OPENSCRIPT

MESSAGES

make
destroy
moved
sized
buttonClick
buttonDown
buttonUp
buttonDoubleClick
buttonStillDown
rightButtonDown
rightButtonUp
rightButtonDoubleClick
mouseEnter
mouseLeave

PROPERTIES

bounds
defaultAllowDrag
defaultAllowDrop
dragImage
drawDirect
fillColor
idNumber
layer
lineEndSize
lineEndStyle
lineStyle
name
noDropImage
notifyAfterMessages
notifyBeforeMessages
object
parent
pattern
position
rgbFill
rgbStroke
script
sharedScript
size
strokeColor
transparent
uniqueName
userProperties
useWindowsColors
vertices
visible

ACTIONS EDITOR

EVENTS

click
double-click
load page
mouse off
mouse over
property change
reset
right-click
trigger
unload page
user event

PROPERTIES

height
left
name
top
visible
width

NOTES

- Draw Objects include angledLine, arc, curve, ellipse, irregularPolygon, line, pie, polygon, rectangle, and roundedRectangle.
- Draw Objects generally get converted to GIF images during a DHTML export.
- You can convert one style of draw object into another by using the Convert Object feature found in the Edit menu.

FIELD

OPENSCRIPT

MESSAGES

make
destroy
enterField
leaveField
moved
sized
textScrolled
buttonClick
buttonDown
buttonUp
buttonDoubleClick
buttonStillDown
rightButtonDown
rightButtonUp
rightButtonDoubleClick
mouseEnter
mouseLeave
keyChar
keyDown
keyUp

PROPERTIES

activated
baselines
borderStyle
bounds
defaultAllowDrag
defaultAllowDrop
dragImage
drawDirect
drawTextDirect
enabled
fieldType
fillColor
fontFace
fontSize
fontStyle
idNumber
indents
layer
name
noDropImage
notifyAfterMessages
notifyBeforeMessages
object
objects
parent
position
rgbFill
rgbStroke
richText
script
scroll
selectedTextlines
sharedScript
singleLine
size
spacing
strokeColor
tabSpacing
tabType
text
textAlignment
textOverflow
textRightOverflow
textUnderflow
transparent
uniqueName
userProperties
useWindowsColors
vertices
visible

ACTIONS EDITOR

EVENTS

click
double-click
key down
key up
key press
load page
property change
reset
right-click
trigger
unload page
user event

PROPERTIES

height
left
name
rgbFill
scroll
text
top
visible
width

NOTES

- A field can only hold 32,000 characters.
- To remove all character level formatting from text contained in a field, set the richText of the field equal to the text of the field.
- A field can only contain one paragraph setting. As such it is not possible to have paragraph one left justified and paragraph two centered.
- A bunch of fields placed in a grid fashion, and grouped together make a great substitute for a Table object.

GROUP

OPENSCRIPT

MESSAGES

make
destroy
moved
sized

PROPERTIES

autoRadioButtons
bounds
dragImage
drawDirect
idNumber
layer
name
noDropImage
notifyAfterMessages
notifyBeforeMessages
object
objects
parent
position
script
sharedScript
size
transparent
uniqueName
userProperties
vertices
visible

ACTIONS EDITOR

EVENTS

click
double-click
load page
mouse off
mouse over
property change
reset
right-click
trigger
unload page
user event

PROPERTIES

enabled
height
left
name
top
visible
width

NOTES

- Radio Buttons contained within a group will behave in a mutually exclusive manner if the autoRadioButtons group property is set to true.
- Setting the ASYMI_ExportAsGraphic user property setting of a group to true will force the entire group to be exported to DHTML as a single image.
- When resizing a group, the objects of the group do not get a sized notification message. Only the group itself gets notified.

HOTWORD

OPENSCRIPT

MESSAGES

make
destroy
moved
sized
buttonClick
buttonDown
buttonUp
buttonDoubleClick
buttonStillDown
rightButtonDown
rightButtonUp
rightButtonDoubleClick
mouseEnter
mouseLeave

PROPERTIES

bounds
defaultAllowDrag
defaultAllowDrop
dragImage
highlight
hotwordStyle
idNumber
invert
name
noDropImage
notifyAfterMessages
notifyBeforeMessages
object
parent
script
sharedScript
text
textOffset
uniqueName
userProperties

ACTIONS EDITOR

EVENTS

click
double-click
load page
property change
right-click
unload page

PROPERTIES

none

NOTES

- The maximum number of hotwords allowed in a single field is 255.
- The parent of a hotword is a Field/RecordField. To determine what hotwords exist in a field you can check the Objects property of the field.
- Hotwords are shown in a style equal to the current hotwordStyle setting of the book. If you set this hotwordStyle book setting to “none”, you can individually color your hotwords.

OPENSCRIPT

MESSAGES

make
destroy
moved
sized
buttonClick
buttonDown
buttonUp
buttonDoubleClick
buttonStillDown
rightButtonDown
rightButtonUp
rightButtonDoubleClick
mouseEnter
mouseLeave

PROPERTIES

allowAuthorActivate
allowReaderActivate
bounds
hasPropertyDialog
idNumber
layer
methods
name
notifyAfterMessages
notifyBeforeMessages
object
parent
position
script
sendToolbookMessages
sharedScript
size
suspendMessages
uniqueName
userProperties
visible

ACTIONS EDITOR

EVENTS

property change
reset
trigger
unload page

PROPERTIES

none

NOTES

- ToolBook 5.0 and lower supported OLE 1.0.
- ToolBook 6.0 and higher support OLE 2.0, but not OLE 1.0.

PAGE

OPENSCRIPT

MESSAGES

All Menu Event Messages
activateInstance
make
destroy
idle
moved
sized
keyChar
keyDown
keyUp
enterPage
leavePage
enterBackground
leaveBackground
enterApplication
leaveApplication
enterBook
leaveBook
enterSystem
leaveSystem

PROPERTIES

defaultAllowDrop
idNumber
imageInvalid
name
notifyObjects
object
objectCount
objects
pageNumber
parent
percentFreeSpace
script
sharedScript
shownBy
skipNavigation
storeImage
storedImages
uniqueName
userProperties

ACTIONS EDITOR

EVENTS

click
double-click
key down
key press
key up
load page
property change
reset
right-click
trigger
unload page
user event

PROPERTIES

height
name
visited
width

NOTES

- Many messages (such as leaveBackground) are typically handled by the programmer at the background or book level, even though the message originates at the page level.
- A page does not have a size. The size of a page is controlled by the size of the background.
- ASYM_BeenHere of a page only gets set when Leaving the page. If you are handling the leavePage message, ensure your Forward the message.

PAINT OBJECT

OPENSCRIPT

MESSAGES

make
destroy
moved
sized
buttonClick
buttonDown
buttonUp
buttonDoubleClick
buttonStillDown
rightButtonDown
rightButtonUp
rightButtonDoubleClick
mouseEnter
mouseLeave

PROPERTIES

bounds
defaultAllowDrag
defaultAllowDrop
dragImage
drawDirect
fillColor
idNumber
layer
lineStyle
name
noDropImage
notifyAfterMessages
notifyBeforeMessages
object
parent
position
rgbFill
rgbStroke
script
sharedScript
size
strokeColor
transparent
uniqueName
useChromakey
userProperties
useWindowsColors
vertices
visible

ACTIONS EDITOR

EVENTS

click
double-click
load page
mouse off
mouse over
property change
reset
right-click
trigger
unload page
user event

PROPERTIES

height
left
name
top
visible
width

NOTES

- Paint Objects can't be resized in ToolBook. If you try to resize a Paint Object you will end up cropping the image instead of resizing it.
- Paint Objects support Transparency - in the same way a GIF image does. Simply set the useChromakey property to true and specify the desired transparent color by setting the fillColor property.
- You can generate a Bitmap resource from a Paint Object by simply pasting the Paint Object into a text field.

PICTURE OBJECT

OPENSCRIPT

MESSAGES

make
destroy
moved
sized
buttonClick
buttonDown
buttonUp
buttonDoubleClick
buttonStillDown
rightButtonDown
rightButtonUp
rightButtonDoubleClick
mouseEnter
mouseLeave

PROPERTIES

bounds
defaultAllowDrag
defaultAllowDrop
dragImage
drawDirect
fillColor
idNumber
layer
lineStyle
name
noDropImage
notifyAfterMessages
notifyBeforeMessages
object
parent
position
rgbFill
rgbStroke
script
sharedScript
size
solidColorsEnabled
strokeColor
uniqueName
userProperties
useWindowsColors
vertices
visible

ACTIONS EDITOR

EVENTS

click
double-click
load page
mouse off
mouse over
property change
reset
right-click
trigger
unload page
user event

PROPERTIES

height
left
name
top
visible
width

NOTES

- Picture Objects tend to paint slower on the screen than a Paint Object will. For this reason, it is recommended that you convert your Picture Objects to Paint Objects where feasible.
- You can convert a Picture Object into a Paint Object. Use the 'convert' button in the Picture Object's property sheet.
- You can generate a Bitmap resource from a Picture Object by simply pasting the Picture Object into a text field.

RECORDFIELD

OPENSCRIPT

MESSAGES

make
destroy
enterRecordField
leaveRecordField
moved
sized
textScrolled
buttonClick
buttonDown
buttonUp
buttonDoubleClick
buttonStillDown
rightButtonDown
rightButtonUp
rightButtonDoubleClick
mouseEnter
mouseLeave
keyChar
keyDown
keyUp

PROPERTIES

activated
baselines
borderStyle
bounds
defaultAllowDrag
defaultAllowDrop
dragImage
drawDirect
drawTextDirect
enabled
fieldType
fillColor
fontFace
fontSize
fontStyle
idNumber
indents
layer
name
noDropImage
notifyAfterMessages
notifyBeforeMessages
object
objects
parent
position
rgbFill
rgbStroke
richText
script
scroll
selectedTextlines
sharedScript
singleLine
size
spacing
strokeColor
tabSpacing
tabType
text
textAlignment
textOverflow
textRightOverflow
textUnderflow
transparent
uniqueName
userProperties
useWindowsColors
vertices
visible

ACTIONS EDITOR

EVENTS

click
double-click
key down
key up
key press
load page
property change
reset
right-click
trigger
unload page
user event

PROPERTIES

height
left
name
rgbFill
scroll
text
top
visible
width

NOTES

- A recordField can only hold 30,000 characters.
- A recordField can only exist on a background.
- In OpenScript, refer to the recordField as if it were on the Page level if you need to access its text properties.
- In OpenScript, if you need to manipulate the recordField properties (except textual properties), refer to the recordField as it exists on the background.

STAGE

OPENSCRIPT

MESSAGES

make
destroy
moved
sized
buttonClick
buttonDown
buttonUp
buttonDoubleClick
buttonStillDown
rightButtonDown
rightButtonUp
rightButtonDoubleClick
mouseEnter
mouseLeave

PROPERTIES

borderWidth
bounds
defaultAllowDrag
defaultAllowDrop
dragImage
drawDirect
fillColor
idNumber
innerBevelWidth
innerBounds
layer
mediaBounds
mediaOpen
mediaSize
name
noDropImage
notifyAfterMessages
notifyBeforeMessages
object
outerBevelWidth
outline
overlayAlias
overlayOpen
parent
position
postEffect
preEffect
readerVisible
rgbFill
rgbStroke
roundedCorners
script
sharedScript
size
stageAnchor
stageSizing
strokeColor
transparent
uniqueName
userProperties
useWindowsColors
vertices
visible

ACTIONS EDITOR

EVENTS

click
double-click
load page
mouse off
mouse over
property change
reset
right-click
trigger
unload page
user event

PROPERTIES

left
top
visible

NOTES

- Although you can place other ToolBook objects on top of or partially overlapping a stage, when a stage plays a media file, the media will always play on top of all ToolBook objects.
- You can add transition effects to media played in a stage by modifying the pre and post effects assigned to the stage object.
- Even if a stage is hidden, you can still play media in the stage and the media will be visible when played.

OPENSCRIPT

MESSAGES

enterMenu
enterWindow
leaveWindow
hidden
menuItemSelected
moved
openWindow
closeWindow
shown
sized

PROPERTIES

alwaysOnTop	failedAuthorPassword	readerStatusBar
alwaysReader	focus	revertFocus
authorStatusBar	hiddenByHideOnReader	rgbMat
autoClose	hideOnDeactivate	rulers
autoShow	hideOnReader	script
autoSize	icon	selectedHotwords
borderStyle	idNumber	selectedText
bounds	imageBuffers	selectedTextState
caption	isOpen	selection
captionBar	lockScreen	sharedScript
caretFontFace	magnification	size
caretFontSize	matColor	state
caretFontStyle	maximumSize	style
caretLocation	menuBar	tile
centerClient	minimumSize	tileOrder
clientHandle	mousePosition	type
clientHandle32	name	uniqueName
clientSize	object	userProperties
currentPage	onBackground	useWindowsColors
defaultClientSize	pageScroll	vertices
defaultPage	parent	visible
defaultPosition	parentHandle	windowHandle
defaultState	parentHandle32	windowHandle32
defaultType	parentWindow	
enabled	position	

ACTIONS EDITOR

EVENTS

property change
reset
trigger
user event

PROPERTIES

none

NOTES

- The viewer that shows your application is viewer id 0 or “mainWindow”.
- The “in viewer” structure can be used in OpenScript to specify that all further OpenScript statement will execute as if targetWindow is your specified viewer.
- Viewers can't be exported to DHTML.
- Need a close button for a popped up viewer? You can close it by providing a button to the user, and having the button execute the following code: close this window

(Add) +

Arithmetic Operators

SYNTAX `<expression> + <expression>`

DESCRIPTION Mathematically adds the two expressions.

NOTES This operator is not used to concatenate strings, use & if your need is to join string segments together.

Although `x = x + 1` can be used to increase the value of `x` by `1`, you can alternately use `increment x`.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that results in a numeric value.

EXAMPLES

```
answer = 4 + 5 + 6
step k from 1 to (total + 5)
if subtotal + 5.5 > maxTotal
total = total + 1
get setValue(k + j)
```

(Subtract) -

Arithmetic Operators

SYNTAX `<expression> - <expression>`

DESCRIPTION Mathematically subtracts the right expression from the left expression.

NOTES Although `x = x - 1` can be used to decrease the value of `x` by `1`, you can alternately use `decrement x`.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that results in a numeric value.

EXAMPLES

```
answer = 4 + 5 - 6
step k from 1 to (total - 5)
if subtotal - 5.5 > maxTotal
total = total - 1
get setValue(k - j)
```

(Comment Character) --

OpenScript Formatting

SYNTAX

```
-- <comment>
<code> -- <comment>
```

DESCRIPTION Indicates a developer comment within OpenScript code.

NOTES Whatever appears to the right of the comment character and before the end of the line is not executed by ToolBook.

If you are putting a comment on the same line as a statement and want to continue the statement on the next line, put the continuation character (\) at the end of the comment, not between the statement and the beginning of the comment.

PARAMETERS

PARAMETER	DESCRIPTION
<comment>	Any comment text.
<code>	Any standard OpenScript code.

EXAMPLES

```
-- This is a comment
val = 50
step k from 1 to 10
    increment val by k -- A comment at the end of a line
    request val
    put val into text of field "apple" -- another comment \
      of page "Lesson"
end
```

(Multiply) *

Arithmetic Operators

SYNTAX `<expression> * <expression>`

DESCRIPTION Mathematically multiplies the two expressions.

NOTES Although `y = x * x` can be used to calculate the value of `x` times itself, you can alternately use `y = x^2`.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<expression>	An expression that results in a numeric value.

EXAMPLES

```
answer = x * y * z

step k from 1 to (total * 5)

if subtotal * 5.5 > maxTotal

get setValue(k * .5)
```

(Divide) /

Arithmetic Operators

SYNTAX `<expression> / <expression>`

DESCRIPTION Mathematically divides the left expression by the right expression.

NOTES Dividing by `0` is not a legal mathematical operation. If your right expression results in a `0`, you will get an error stating: `Divide By 0`.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<expression>	An expression that results in a numeric value.

EXAMPLES

```
answer = 4 * 5 / 6
step k from 1 to (total / 2)
if subtotal / units > maxTotal
get setValue(k / j)
```

(Continuation Character) \

OpenScript Formatting

SYNTAX `<code> [-- <comment>] \`

DESCRIPTION In the Script editor, you can split an OpenScript statement across more than one line by using the backslash character (\) as a continuation character at the end of a line.

PARAMETERS

PARAMETER	DESCRIPTION
`<comment>`	Any comment text.
`<code>`	Any standard OpenScript code.

EXAMPLES

```
val = 50
step k from 1 to 10
    increment val by (text of field "produce" of page "Lesson" of \
      background "Farming" + k)
    request val
    put val into text of field "apple" -- a comment can go here \
      of page "Lesson"
end
```

(Exponentiate) ^

Arithmetic Operators

SYNTAX `<expression> ^ <expression>`

DESCRIPTION Using exponentiation, raises the left expression to the power of the right expression.

For example `x = 9^5` is the same as `x = 9 * 9 * 9 * 9 * 9`

NOTES Be careful when grouping a series of exponentiations. The evaluation order is not what you might expect.

`x = 2^3^4` is not evaluated from left to right as in `x = (2^3)^4` but is instead evaluated from right to left as in `x = 2 ^ (3^4)`.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that results in a numeric value.

EXAMPLES

```
answer = 4 ^ 5 ^ 6
```

```
step k from 1 to (total ^ 2)
```

```
x = y ^ z
```

(Equal) =

Arithmetic Operators

SYNTAX `<expression> = <expression> [as <type>]`

DESCRIPTION Compares the expressions and returns `true` if the left expression and the right expression are equivalent. Returns `false` if the expressions are not equivalent.

The expressions can be compared `as text`, `as numbers`, `as dates`, or `as names`.

The `is` operator works identically to the = operator.

NOTES The = symbol can also be used to assign a value, instead of comparing a value, the second entry of = for full details.

In OpenScript, string comparisons are case insensitive.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<expression>	Any expression that yields a numeric value or a value appropriate to the value of the <type> parameter.
<type>	number, text, date, or name The default value of the <type> parameter is number, resulting in numeric comparison. If the <type> parameter is text, the two expressions are compared alphabetically, according to the ANSI values of the characters. If the <type> parameter is date, the expressions are compared as dates according to the current value of the sysDateFormat property.

EXAMPLES

```
while userName = "admin"

if password = "12345"

put (age = 65) into checked of button "retire"
```

(Equals) =

Assigning Values

SYNTAX

```
<container> [of <object>] = <value | container> [of <object>]
```

DESCRIPTION

Assigns a value to a specified property or other container. The assignment is made by first resolving the value on the right of the = symbol, and then assigning that value to the container specified on the left side of the = symbol.

NOTES

You can also use the set command to assign a value to a container.

If you attempt to use the = assignment operator with a property name that is not a standard property, ToolBook will search the object hierarchy for a corresponding to set handler for that property name. If no such handler exists, ToolBook then treats the name as a user property and sets its value.

ACTIONS EDITOR

The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<container>	A valid container such as a variable, or a property.
<object>	A reference to an object.
<value | container>	A setting for a property or container.

EXAMPLES

```
x = 5

text of field "list" of page "text" = caption of button "start"

item 2 of position of self = 16
```

(Concatenate) &

String Operators

SYNTAX

```
<expression> & <expression>
```

DESCRIPTION

Joins two string values together into a larger single string value.

NOTES

The ampersand (&) also has a totally different purpose. It makes a character in a button's caption a mnemonic access character.

ACTIONS EDITOR

The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<expression>	Any Expression.

EXAMPLES

```
fullName = firstName & space & lastName

if partNum & binNum = "E1119B12"
    request "Sorry, this part is no longer stocked"
end
```

(Concatenate with Space) &&

String Operators

SYNTAX

```
<expression> && <expression>
```

DESCRIPTION

Joins two string values together into a larger single string value and put a space between the two strings.

NOTES

The ampersand (&) also has a totally different purpose. It makes a character in a button's `caption` a `mnemonic` access character.

ACTIONS EDITOR

The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<expression>	Any Expression.

EXAMPLES

```
fullName = firstName && lastName

if partNum && binNum = "E1119 B12"
    request "Sorry, this part is no longer stocked"
end
```

(Not Equal) <>

Logic Operators

SYNTAX

```
<expression> <> <expression> [as <type>]
```

DESCRIPTION

Compares the expressions and returns `true` if the left expression and the right expression are different. Returns `false` if the expressions are the same.

The expressions can be compared `as text`, `as numbers`, `as dates`, or `as names`.

The `is not` operator works identically to the <> operator.

NOTES

In OpenScript, string comparisons are case insensitive.

ACTIONS EDITOR

The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<expression>	Any expression that yields a numeric value or a value appropriate to the value of the <type> parameter.
<type>	number, text, date, or name The default value of the <type> parameter is number, resulting in numeric comparison. If the <type> parameter is text, the two expressions are compared alphabetically, according to the ANSI values of the characters. If the <type> parameter is date, the expressions are compared as dates according to the current value of the sysDateFormat property.

EXAMPLES

```
if password <> "12345"
    request "you don't have access to this file"
end

put (age <> 65) into checked of button "notRetire"
```

(Less Than) <

SYNTAX `<expression> < <expression> [as <type>]`

DESCRIPTION Compares the expressions and returns `true` if the left expression is less than the right expression. Otherwise returns `false`.

The expressions can be compared `as text`, `as numbers`, `as dates`, or `as names`.

NOTES In OpenScript, string comparisons are case insensitive.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	Any expression that yields a numeric value or a value appropriate to the value of the `<type>` parameter.
`<type>`	`number`, `text`, `date`, or `name` The default value of the `<type>` parameter is `number`, resulting in numeric comparison. If the `<type>` parameter is `text`, the two expressions are compared alphabetically, according to the `ANSI` values of the characters. If the `<type>` parameter is `date`, the expressions are compared as dates according to the current value of the `sysDateFormat` property.

EXAMPLES

```
if currentAge < 65
    go to page "cantRetireYet"
end
```

(Greater Than) >

SYNTAX `<expression> > <expression> [as <type>]`

DESCRIPTION Compares the expressions and returns `true` if the left expression is greater than the right expression. Otherwise returns `false`.

The expressions can be compared `as text`, `as numbers`, `as dates`, or `as names`.

NOTES In OpenScript, string comparisons are case insensitive.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	Any expression that yields a numeric value or a value appropriate to the value of the `<type>` parameter.
`<type>`	`number`, `text`, `date`, or `name` The default value of the `<type>` parameter is `number`, resulting in numeric comparison. If the `<type>` parameter is `text`, the two expressions are compared alphabetically, according to the `ANSI` values of the characters. If the `<type>` parameter is `date`, the expressions are compared as dates according to the current value of the `sysDateFormat` property.

EXAMPLES

```
if currentAge > 65
    go to page "retire"
end
```

(Less Than or Equal) <=

SYNTAX

```
<expression> <= <expression> [as <type>]
```

DESCRIPTION

Compares the expressions and returns `true` if the left expression is less than or equal to the right expression. Otherwise returns `false`.

The expressions can be compared `as text`, `as numbers`, `as dates`, or `as names`.

NOTES

In OpenScript, string comparisons are case insensitive.

ACTIONS EDITOR

The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	Any expression that yields a numeric value or a value appropriate to the value of the <type> parameter.
`<type>`	`number`, `text`, `date`, or `name` The default value of the `<type>` parameter is `number`, resulting in numeric comparison. If the `<type>` parameter is `text`, the two expressions are compared alphabetically, according to the `ANSI` values of the characters. If the `<type>` parameter is `date`, the expressions are compared as dates according to the current value of the `sysDateFormat` property.

EXAMPLES

```
if text of field "Date" <= sysDate as date
    text of field "Access_dates" = sysDate
end
```

(Greater Than or Equal) >=

SYNTAX

```
<expression> >= <expression> [as <type>]
```

DESCRIPTION

Compares the expressions and returns `true` if the left expression is greater than or equal to the right expression. Otherwise returns `false`.

The expressions can be compared `as text`, `as numbers`, `as dates`, or `as names`.

NOTES

In OpenScript, string comparisons are case insensitive.

ACTIONS EDITOR

The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	Any expression that yields a numeric value or a value appropriate to the value of the <type> parameter.
`<type>`	`number`, `text`, `date`, or `name` The default value of the `<type>` parameter is `number`, resulting in numeric comparison. If the `<type>` parameter is `text`, the two expressions are compared alphabetically, according to the `ANSI` values of the characters. If the `<type>` parameter is `date`, the expressions are compared as dates according to the current value of the `sysDateFormat` property.

EXAMPLES

```
if sysDate >= text of field "Date" as date
    text of field "Access_dates" = sysDate
end
```

a, an

Articles and Prepositions

DESCRIPTION Used with the `draw` command to make statements easier to read. This terms is optional in every context in which it can be used.

EXAMPLES

```
draw a line from 0,0 to 144, 166
draw an arc from 0,0 to 300,300 to 0,600
```

abs()

Arithmetic Values

SYNTAX

```
abs(<number>)
```

DESCRIPTION Returns the absolute value of a number, a number without its sign. This means that the return value from this function will never be a negative number.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<number>`	An expression that results in a number.

EXAMPLES

```
x = 6
y = 10
z = abs(x – y) -- results in 4 even though 6 - 10 is really -4
```

acos()

Trigonometric Values

SYNTAX

```
acos(<number>)
```

DESCRIPTION Returns the arccosine of a number. The arccosine is the angle whose cosine is number. The returned angle is given in radians in the range `0` (zero) to `pi`.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<number>`	An expression that yields a number in the range `-1` to `1`.

EXAMPLES

```
x = acos(-0.5)           -- equals 2.094395 (2*pi/3 radians)
x = acos(-0.5)*180/pi    -- equals 120 (degrees)
```

activate

Viewer Command

SYNTAX

```
activate <viewer reference>
```

DESCRIPTION Activates a viewer by bringing it in front of other viewers and making its contents available to the user.

NOTES Activating a viewer using the `activate` command is identical to setting the `focusWindow` system property for that viewer.

A viewer is also activated when a user clicks it, or when it is shown using the `show <viewer reference>` command.

PARAMETERS

PARAMETER	DESCRIPTION
`<viewer reference>`	A valid reference to a viewer object. To find a viewer's reference based on its window handle, use the `windowRefFromHandle()` function.

EXAMPLES

```
activate viewer "help"
```

activated

Property

DESCRIPTION A field or recordfield property that specifies whether a field or recordfield can be edited (typed into), or whether it can receive mouse event messages at `Reader` level.

NOTES You can get or set this property.

If `activated` is `true`, the field or recordfield cannot be edited. Clicking the field sends mouse event messages to the field or to its hotwords.

If `activated` is `false`, the field or recordfield can be edited, and the cursor changes to an I-beam when it enters that field.

VALUES `True` or `false`.

The default is `false`.

When you set a recordfield's `fieldType` property to `singleSelect` or `multiSelect`, its `activated` property is automatically set to `true`.

EXAMPLES

```
-- Allows typing in a field when a user clicks it
-- with the right mouse button
to handle rightButtonUp
    if object of target is "field"
        activated of target = false
    end
end
```

activateInstance

Event Message

DESCRIPTION Sent to the current page when an instance of ToolBook becomes active.

NOTES If an instance is activated because a user clicks the menu bar or a scroll bar, ToolBook does not send the `activateInstance` message until the user releases the mouse button.

If the user chooses a command from the menu bar, ToolBook does not send the `activateInstance` message until after it completes the command.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle activateInstance
    sysHistoryRecord = false
    forward
end
```

activeCacheFile

Property

DESCRIPTION A book property that specifies whether a cache file is being used by the book.

NOTES This is a read only property and cannot be set.

Use this property to determine if the application is running with the correct `cache file`.

For example, you may want to write a script to verify that a book is using a cache file. If it isn't, you can then prompt the user with instructions on how to install that cache file.

VALUES `None`, `temporary`, or the file name of a permanent cache file in use.

EXAMPLES

```
-- Verifies that a cache file is used when the book opens
to handle enterBook
    if activeCacheFile of this book = "none"
        request "Please install a cache file."
    end
end
```

activeWindowHandle

System Property

DESCRIPTION A system property that specifies the `16-bit` window handle of the currently active window.

NOTES You can get or set this property.

The window can be either a ToolBook window such as a viewer, or a window from another application.

VALUES A `16-bit` number assigned by Windows that specifies the window handle of the active window.

If you attempt to set the value to an invalid window handle, the set operation fails and the currently active window remains active.

Only one window can be active at any time.

The value of `activeWindowHandle` is independent of the ToolBook instance (because it is assigned by Windows) and can be any top-level or popup window on the Windows desktop.

It does not necessarily need to be a ToolBook viewer or the focus window.

EXAMPLES

```
activeWindowHandle = windowHandle of viewer "help"
```

activeWindowHandle32

System Property

DESCRIPTION A system property that specifies the `32-bit` window handle of the currently active window.

NOTES You can get or set this property.

The window can be either a ToolBook window such as a viewer, or a window from another application.

VALUES A `32-bit` number assigned by Windows that specifies the window handle of the active window.

If you attempt to set the value to an invalid window handle, the set operation fails and the currently active window remains active.

Only one window can be active at any time.

The value of `activeWindowHandle32` is independent of the ToolBook instance (because it is assigned by Windows) and can be any top-level or popup window on the Windows desktop.

It does not necessarily need to be a ToolBook viewer or the focus window.

EXAMPLES

```
activeWindowHandle32 = windowHandle32 of viewer "help"
```

add menu

Menu Command/Function

SYNTAX

```
add menu <menu name> [alias <alias>] [in <menu reference>
  [in <menu reference>] ... ] [position <position>] [at <level>]
  [with helpText <help text>]
```

DESCRIPTION

Adds a `menu` to the menu bar of the target window. The menu appears to the right of all other menus unless a position is specified. If `<help text>` is specified with the menu, that text appears in the status bar when the user chooses the menu.

NOTES

You can create a `submenu` by adding it to an existing menu using the in `<menu reference>` parameter. You can use this parameter multiple times to refer to a cascading `menu` with multiple submenus. Submenus can only be added to menus, not menu items.

PARAMETERS

PARAMETER	DESCRIPTION
`<menu name>`	The name of the menu as it is to appear on the menu bar. The menu name must include at least one alphabetic character or underscore character, and can be up to `60` characters long. Enclose the menu name in quotation marks if it contains spaces or punctuation. To define a mnemonic access character for the menu, place an ampersand (`&`) before the character that is to serve as the access key.
`<alias>`	A word that specifies the `message` to be sent when the menu is chosen. When the user chooses a menu with an `alias`, the `alias` is sent as the message rather than the menu name. This parameter is optional.
`<menu reference>`	A valid reference to an existing `menu` or `submenu` in the target window. If more than one menu shares the same reference, ToolBook acts on the first menu from the left on the menu bar.
`<position>`	A positive integer indicating the placement of the menu. For example, position 3 places the menu in the third menu position from the left on the menu bar. If a menu already holds that position, that menu and all menus to its right shift right to accommodate the new menu. The `Help` menu in a book's default menu bar is always in the *last position*.
`<level>`	`Author`, `reader`, or `both`; if no level is specified, the default is the current working level when the add menu command is executed.
`<help text>`	A string of text that appears in the status bar when the menu is selected. If the parameter is omitted or `null`, no text appears in the status bar. This parameter is optional.

EXAMPLES

```
add menu "Books" position 3 at Reader with helpText \
  "Provides controls for books"

in viewer "Browser"
    add menu "Options"
end in

add menu "Re&ferences" alias "Refs" in menu "topics" \
  at Reader with helpText "Shows the References screen"
```

add menuItem

Menu Command/Function

SYNTAX

```
add menuItem <menu item> [alias <alias>] to menu <name | alias>
  [in <menu reference> [in <menu reference>] ... ] [position
  <position>] [at <level>] [with helpText <help text>]
```

DESCRIPTION Adds a `menu item` to a `menu` on the menu bar of the target window. If `<help text>` is specified with the menu item, that text appears in the status bar when the user chooses that menu item.

NOTES You can add a menu item to a submenu by using the in `<menu reference>` parameter. You can use this parameter multiple times to refer to a cascading menu with multiple submenus. Submenus can only be added to menus, not menu items.

ToolBook sends a message `alias` to the current page when a built-in menu item is chosen by the user. If the `alias` refers to a built-in ToolBook menu event message, ToolBook uses any automatic behavior associated with its menu item; for example, `Cut` is disabled when there is no selection to act on.

For ToolBook to respond to a message that is not built in, you must write a `handler` for the message and place it in the page script or higher in the object hierarchy. If no handler exists for the message, nothing happens when the user selects the new menu item.

Leading numerals (menu items that start with a number) and all non-alphanumeric characters, such as spaces and punctuation, are omitted from the message that is sent when the user chooses the menu item. For example, choosing `Enter Costs ...` sends the message `enterCosts.`

The maximum number of menu items is `128`.

PARAMETERS

PARAMETER	DESCRIPTION
`<menu item>`	The name of a menu item up to `60` characters in length. The menu name must include at least one alphabetic character or underscore character. Enclose the menu item name in quotation marks if it contains spaces or punctuation. To define a mnemonic access character for a menu item, place an ampersand (`&`) before the character in the menu item's name that is to serve as the access key. To create a `separator bar` between menu items, use `null` for `<menu item>`.
`<alias>`	A word that specifies the message to be sent when the menu item is chosen. When the user chooses a menu item with an alias, the alias is sent as the message rather than the name. This parameter is optional.
`<name | alias>`	The name of the `menu` as it appears on the menu bar, or the `alias` assigned to the menu.
`<menu reference>`	A valid reference to an existing `menu` or `submenu` in the target window. If more than one menu shares the same reference, ToolBook acts on the first menu from the left on the menu bar.
`<position>`	A positive integer that indicates the placement of the menu item on the menu. For example, position 3 means the third item on the menu. If a menu item already holds that position, that menu item and all others below it shift down to accommodate the new menu item. If an invalid position is specified (the position does not exist), ToolBook places the menu item first or last on the menu, depending on which location is closer to the specified position. If no position is specified, the menu item is placed last on the menu.
`<level>`	`Author`, `Reader`, or `both`; if a level is not specified, the default is the current working level when the `add menuItem` command is executed.
`<help text>`	A string of text that appears in the status bar when the menu item is selected. If the parameter is omitted or `null`, no text appears in the status bar. This parameter is optional.

EXAMPLES

```
add menuItem "List" to menu "Books" position 1 with helpText \
  "Creates a list of books available"

add menuItem "&Calendar..." to menu "Books" at both

add menuItem "Glossary" to menu "&Multimedia" in menu "Help" \
  at author with helpText "Help on multimedia functions"
```

add sysBook

System Book Related Commands

SYNTAX `add sysBook [<file name>]`

DESCRIPTION Adds a bound system book to the book.

To determine which system books are currently loaded, check the `sysBooks` system variable.

To determine which system books have been `bound` to the current book, call `queryAddedExtension()` and search through the returned stack for system books, or open the `Bound System Books` dialog box.

NOTES A system book can be added in two ways. First is via the bound system books mechanism, and secondly is by adding the system book reference to the `sysBooks` system property.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	An optional parameter that specifies the name of the file to be added as a bound system book. If the file is not in the search path, then a fully qualified path and file name are required. If `<file name>` is omitted, the default response is to open the `Bound System Books` dialog box.

EXAMPLES

```
add sysBook "MYSBK.SBK"
```

after

Articles and Prepositions

DESCRIPTION Used with various OpenScript commands such as `pop`, `push`, `insert graphic` to indicate that a value should follow other values.

EXAMPLES

```
put "The End" after text of field "story"
```

```
insert graphic bitmap "Fog" after text of focus
```

```
pop mylist after goodList
```

align

Alignment Command

SYNTAX `align <type>`

DESCRIPTION Aligns selected objects according to the alignment option you specify in the `<type>` parameter.

Using this command is similar to choosing `Align` from the `Draw` menu. If no objects are selected, ToolBook displays the `Execution Suspended` message.

PARAMETERS

PARAMETER	DESCRIPTION
`<type>`	`Left`, `right`, `top`, `bottom`, `horizontal`, or `vertical`.

EXAMPLES

```
select all button
align top
```

allowAuthorActivate

Property

DESCRIPTION Controls whether a control can be activated at `Author` level.

NOTES Applies ActiveX controls and an OLE servers

VALUES If `true`, you can double-click the object in `Author` mode and edit it in place.

The default is `true` for OLE servers and `false` for ActiveX controls.

EXAMPLES
```
allowAuthorActivate of tList "mainList" = false
```

allowReaderActivate

Property

DESCRIPTION Controls whether a control can be activated at `Reader` level.

NOTES Applies to OLE servers.

For ActiveX controls the property is always `true`. If you set the property to `false`, it will return to `true`, and the object will continue to activate in `Reader` mode.

VALUES If `true`, you can double-click the object in `Reader` mode and edit it in place.

The default is `true`.

EXAMPLES
```
allowReaderActivate of grfx "mainList" = false
```

alwaysOnTop

Property

DESCRIPTION A property of a viewer that specifies whether a viewer appears on top of all other windows on the desktop when it is shown.

A window that is always on top is displayed above all other windows in the system, including popup windows and windows from other applications.

NOTES You can get or set this property.

If several windows on the desktop are designated to always be on top, they are layered based on which window was most recently active, with the currently active window on top.

Use the `alwaysOnTop` property of a viewer to create a window that is always visible, regardless of what application is currently active.

For example, you might display a Help application in a viewer that always appeared on top of other windows. The user could gain instant access to the Help application by clicking the viewer.

VALUES `True` or `false`.

The default is `false`.

If `true`, the viewer appears on top of all other windows when it is shown.

If `false`, the viewer is layered with other windows.

EXAMPLES
```
alwaysOnTop of viewer "help" = true
show viewer "help"
```

alwaysReader

Property

DESCRIPTION A property of a viewer that specifies whether a viewer operates at `Reader` level even when ToolBook is at `Author` level.

NOTES You can get or set this property for all viewers except the ToolBook Main window.

If `alwaysReader` is `true`, the viewer runs at `Reader` level even if the system is at `Author` level.

You can use the `alwaysReader` property to create custom authoring tools.

VALUES `True` or `false`.

The default is `false`.

If `true`, the viewer operates continuously at `Reader` level, regardless of the system setting.

If `false`, the viewer operates at the current system level determined by the `sysLevel` setting.

EXAMPLES
```
alwaysReader of viewer "help" = true
```

and

Logic Operators

SYNTAX
```
<expression> and <expression>
```

DESCRIPTION Returns `true` if the left expression and right expression both evaluate to `true`. Otherwise returns `false`.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	Any expression.

EXAMPLES
```
if score > 85 and attempts < 3
    request "you passed!"
end
```

angledLine

Object Type

SYNTAX
```
draw angledLine from <location> to <location> to <location>...
```

DESCRIPTION The object type name for an `angled line`.

NOTES An angled line's parent can be the page, the background, or a group.

The `bounds` and `vertices` properties of an angled line do not have the same values. An angled line's `bounds` are the same as the location of the object's selection handles, while each `vertex` is a list of two numbers that give the location of the angled line's vertices from the first angle drawn to the last angle drawn.

Angled lines cannot be filled with a color or pattern.

annuityFactor()

Financial Values

SYNTAX
```
annuityFactor(<rate>,<periods>)
```

DESCRIPTION A financial function that returns a factor of the present value of an ordinary annuity to the annuity payment. The present value of a loan, divided by the annuity factor, yields the payment needed each period to pay off the loan.

The formula used for the calculation is: `(1 - (1 + <rate>)-<periods>)/<rate>`

PARAMETERS

PARAMETER	DESCRIPTION
`<rate>`	Any expression that yields a number representing the interest rate per period expressed as a decimal.
`<period>`	Any expression that yields a number representing the number of periods over which the value of the annuity is calculated and compounded.

EXAMPLES
```
-- Calculates payments for $2000 loan at 10% interest for 12 months
x = carLoanAmt/annuityFactor(periodRate,periods)
```

DESCRIPTION The ANSI character set.

0	*	51	3	102	f	153	™	204	Ì
1	*	52	4	103	g	154	š	205	Í
2	*	53	5	104	h	155	›	206	Î
3	*	54	6	105	i	156	œ	207	Ï
4	*	55	7	106	j	157	***	208	Ð
5	*	56	8	107	k	158	***	209	Ñ
6	*	57	9	108	l	159	Ÿ	210	Ò
7	*	58	:	109	m	160		211	Ó
8	**	59	;	110	n	161	¡	212	Ô
9	**	60	<	111	o	162	¢	213	Õ
10	**	61	=	112	p	163	£	214	Ö
11	*	62	>	113	q	164	¤	215	×
12	*	63	?	114	r	165	¥	216	Ø
13	**	64	@	115	s	166	¦	217	Ù
14	*	65	A	116	t	167	§	218	Ú
15	*	66	B	117	u	168	¨	219	Û
16	*	67	C	118	v	169	©	220	Ü
17	*	68	D	119	w	170	ª	221	Ý
18	*	69	E	120	x	171	«	222	Þ
19	*	70	F	121	y	172	¬	223	ß
20	*	71	G	122	z	173	-	224	à
21	*	72	H	123	{	174	®	225	á
22	*	73	I	124	|	175	¯	226	â
23	*	74	J	125	}	176	°	227	ã
24	*	75	K	126	~	177	±	228	ä
25	*	76	L	127	***	178	²	229	å
26	*	77	M	128	***	179	³	230	æ
27	*	78	N	129	***	180	´	231	ç
28	*	79	O	130	‚	181	µ	232	è
29	*	80	P	131	ƒ	182	¶	233	é
30	*	81	Q	132	„	183	·	234	ê
31	*	82	R	133	…	184	¸	235	ë
32	**	83	S	134	†	185	¹	236	ì
33	!	84	T	135	‡	186	º	237	í
34	"	85	U	136	ˆ	187	»	238	î
35	#	86	V	137	‰	188	¼	239	ï
36	$	87	W	138	Š	189	½	240	ð
37	%	88	X	139	‹	190	¾	241	ñ
38	&	89	Y	140	Œ	191	¿	242	ò
39	'	90	Z	141	***	192	À	243	ó
40	(	91	[	142	***	193	Á	244	ô
41	)	92	\	143	***	194	Â	245	õ
42	*	93	]	144	***	195	Ã	246	ö
43	+	94	^	145	'	196	Ä	247	÷
44	,	95	_	146	'	197	Å	248	ø
45	-	96	`	147	"	198	Æ	249	ù
46	.	97	a	148	"	199	Ç	250	ú
47	/	98	b	149	•	200	È	251	û
48	0	99	c	150	–	201	É	252	ü
49	1	100	d	151	—	202	Ê	253	ý
50	2	101	e	152	˜	203	Ë	254	þ
								255	ÿ

* No conversion for this character

** Values 8, 9, 10, 13, and 32 convert to backspace, tab, linefeed, carriage return, and space, respectively.

*** Values 127-129, 131-144, and 157-158 vary depending on the font manufacturer.

ansiToChar()

String Functions

SYNTAX `ansiToChar(<number>)`

DESCRIPTION Converts a `ANSI` numeric value into a character.

RETURNS Results in a single character being returned by the function.

NOTES You can also convert a character back into an `ANSI` value by using the `charToAnsi()` function.

PARAMETERS

PARAMETER	DESCRIPTION
`<number>`	An expression that results a whole number in the range of `0 – 255`. If you pass a number larger than `255`, ToolBook will perform a `mod` operation it to bring the number back down under the `255` maximum range. Essentially the value of `<number>` goes through this operation: `<number> = <number> mod 255`. If you pass a negative number, ToolBook will generate an `Execution Suspend` error message stating that your number is outside of the allowed range, and sets `sysErrorNumber` to `8018`.

EXAMPLES

```
-- populate a string with A – Z separated by TABs
str = null
-- step through all 26 letters of the alphabet
step k from 1 to 26
    -- since ANSI 65 is where the letter A starts, pad with 64.
    put ansiToChar(k +64) and TAB after str
end
```

arc

Object Type

SYNTAX `draw arc from <location> to <location> to <location>`

DESCRIPTION The object type name for an `arc`.

NOTES An arc's parent can be the page, the background, or a group.

The `bounds` and `vertices` properties of an arc do not have the same values. The `bounds` are the same as the location of the object's selection handles.

The four values for the `vertices` represent the `x,y` coordinates of the arc's beginning and end points. The arc sweeps between the imaginary radii drawn from the center of the arc to its vertices.

argCount

Special Variable

DESCRIPTION A special local variable that contains the number of parameters passed to the currently executing handler or function.

VALUE A non-negative integer equal to the number of parameters passed to the current handler.

NOTES The `argCount` variable is most often used to process parameters passed with user-defined functions, properties, and messages.

This is very handy for processing a list of parameters when you don't know how many parameters will be passed.

EXAMPLES

```
to handle buttonClick
    request mySum(1,2,3,4,5,6,7,8,9)
    request mySum(20,30)
end

to get mySum
    ttl = 0
    step k from 1 to argCount
        increment ttl by item k of argList
    end
    return ttl
end
```

argList

Special Variable

DESCRIPTION A special local variable that contains a list of the parameters passed to the currently executing handler or function.

VALUE A list of the values of all parameters passed to the current handler/function.

If ToolBook doesn't pass any parameters to the current handler, the value is `null`.

NOTES This is very handy for processing a list of parameters when you don't know how many parameters will be passed.

EXAMPLES

```
to handle buttonClick
    request mySum(1,2,3,4,5,6,7,8,9)
    request mySum(20,30)
end

to get mySum
    ttl = 0
    step k from 1 to argCount
        increment ttl by item k of argList
    end
    return ttl
end
```

argument

Special Variable

SYNTAX `argument <expression>`

DESCRIPTION Gets the value of a parameter passed to the currently executing handler.

NOTES A parameter can consist of more than one item, so argument *n* and item *n* of `argList` (where `n <=` the value of `argCount`) can return different values.

This is very handy for processing a list of parameters when you don't know how many parameters will be passed.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that evaluates to a positive integer.

EXAMPLES

```
to handle buttonClick
    request calculatedValue(1,5,9,8)
end

to get calculatedValue
    if argument 3 = 9
        return "Invalid Sequence"
    end
    ttl = 0
    step k from 1 to argCount
        increment ttl by item k of argList
    end
    return ttl
end
```

asin()

Trigonometric Values

SYNTAX

```
asin(<number>)
```

DESCRIPTION

Returns the arcsine of a number. The arcsine is the angle whose sine is number. The returned angle is given in radians in the range -pi/2 to pi/2.

ACTIONS EDITOR

The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<number>	An expression that yields a number in the range -1 to 1.

EXAMPLES

```
x = asin(-0.5)          -- equals -0.5236 (-pi/6 radians)
x = asin(-0.5)*180/pi   -- equals -30 (degrees)
```

ask

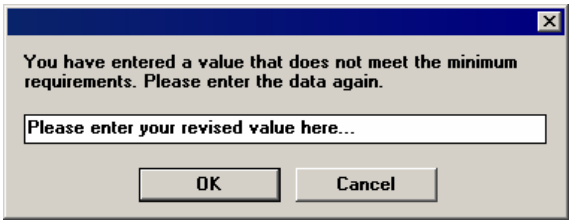
Dialog Box

SYNTAX

```
ask <question> [with <default answer>]
```

DESCRIPTION

Displays a dialog box containing a question, a box where the user can type an answer, and the OK and Cancel buttons.



RETURNS

The user's response returns a value that is placed into IT when the dialog box is dismissed.

If the user clicks OK or presses Enter, IT is set to the string entered in the box.

If the user clicks Cancel, presses Esc, or closes the dialog box with the Control-menu box, IT is set to null and sysError is set to cancel.

NOTES

This dialog box is very limited in functionality. It is recommended that you use ASYM_Ask() instead, which has many more features to offer.

PARAMETERS

PARAMETER	DESCRIPTION
<question>	Any string expression. The amount of text to be display is limited to 3 textlines; more will be truncated.
<default answer>	Any string expression.

EXAMPLES

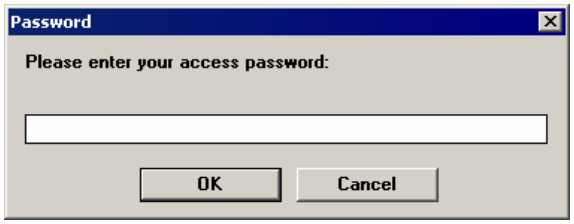
```
ask "Please enter you name:" with "type your entry..."
text of field "name" = IT
```

ask password

Dialog Box

SYNTAX `ask password <question>`

DESCRIPTION Displays a dialog box containing a question, a box where the user can type an answer, and the `OK` and `Cancel` buttons.



RETURNS The user's response returns a value that is placed into `IT` when the dialog box is dismissed. If the user clicks `OK` or presses `Enter`, `IT` is set to the string entered in the box. If the user clicks `Cancel`, presses `Esc`, or closes the dialog box with the Control-menu box, `IT` is set to `null` and `sysError` is set to `cancel`.

NOTES Use an `ask password` statement to control user access to a book. When a user types a password in the Password dialog box, ToolBook encrypts the answer. You can then compare the user's encrypted password with a list of correct passwords (stored in `sysPasswords`), or you can add the user's response to the list of acceptable passwords.

To specify a list of encrypted passwords for ToolBook to check before requesting a password from the user, set the `sysPasswords` property to a string of up to ten encrypted passwords, each on a separate textline.

PARAMETERS

PARAMETER	DESCRIPTION
`<question>`	Any string expression. The amount of text to be display is limited to 3 textlines; more will be truncated.

EXAMPLES

```
ask password "Please enter your access password:"
```

ASYM_AddFileExtension()

TB89R.SBK File Function (Filename)

SYNTAX `ASYM_AddFileExtension(<file name>,<file extension>)`

DESCRIPTION This function can be used to add a file extension to a file name that doesn't already have one.

RETURNS The file name with the extension, if the file name doesn't already have an extension.

NOTES The function examines `<file name>` to determine if it already has an extension. If it does have an extension, the function does nothing. If it doesn't have an extension, it adds the file extension specified in `<file extension>`.

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	The file name you want to add a file extension to.
`<file extension>`	The file extension you want to add to the filename.

EXAMPLES

```
-- Create a log filename based on the user's name
x = fName & lName
fileName = ASYM_AddFileExtension(x,"log")
```

SYNTAX

```
ASYM_AddHyperlink(<object>,<link name>,<page>,<transition>,"jump",
                  <cursor>,<isURL>,<vRef>)

ASYM_AddHyperlink(<object>,<link name>,<page>,<style>,"popup",
                  <cursor>,<isURL>,<vRef>)
```

DESCRIPTION

Assigns a hyperlink to an object.

RETURNS

If successful, the function returns `true`, otherwise it returns `false` and sets `sysError` to a value that indicates the nature of the error.

If an existing hyperlink with the same link name exists, it is replaced by the new hyperlink.

NOTES

This function is provided by the TB89HYP.SBK. To successfully use this function in Runtime, you must first add the TB89HYP.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89HYP.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89HYP.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<object>`	The name of an object.
`<link name>`	The name of the hyperlink to serve as a trigger event, or the standard trigger event `buttonClick`.
`<page>`	A valid reference to the destination page for the hyperlink, or one of these values: "`<next>`", "`<previous>`","`<first>`","`<last>`", or "`<back>`".

`<style>`	An optional window style setting for a popup hyperlink, or `null`. If `null`, the default is "`shadowAutoClose`".
	`"shadow"`
	`"shadowAutoClose"`
	`"thickFrame"`
	`"thinFrame"`
	`"dialogFrame"`
	If you set `<style>` to `shadow`, you must include a script to close the window. For other styles, the interface provides a method for the user to close the window without further scripting.
	A simple script to close the current window would be:
	`to handle buttonClick`
	`  close this window`
	`end`

`<transition>`	An optional transition effect setting for a `jump` hyperlink. A string of options that describe the transition effect used in navigating to the page. The value can be `null`. The effect string uses the following syntax: `"<effect> <destination> <[speed] speed value>"` For example: `"slide left speed 750"` defines a `jump` transition. For a list of valid effect strings, see `transition`.
`<cursor>`	An optional `sysCursor` number, or `null`.
`<isURL>`	An optional parameter specifying if the page parameter is a `URL`. A value of `true` indicates that the page parameter is a `URL`. A value of `false` indicates that the page parameter is not a `URL`. The default is `false`.
`<vref>`	Optional reference to the viewer in which the hyperlink is to be shown. If `null`, the viewer containing the object is used. Specify the main window using the value `<mainWindow>`, or specify the target window by using the value `<targetWindow>`.

EXAMPLES

```
get ASYM_AddHyperlink(self,"buttonClick",page 1 of book "book2.tbk", \
                      "dialogFrame","popup",4,false)

get ASYM_AddHyperlink(self,"buttonClick",page 2,"slide in top", \
                      "jump",NULL,false)

get ASYM_AddHyperlink(button "First","buttonClick","<first>","fade", \
                      "jump",4,false,this window)

get ASYM_AddHyperlink(button "First","buttonClick",          \
                      "http://www.xyzcorp.com",NULL,"popup", \
                      NULL,true)
```

ASYM_AddString()

TB89R.SBK String Functions

SYNTAX

`ASYM_AddString(<string name>,<string text>)`

DESCRIPTION

Adds or replaces a `String Resource` entry in the `String Resource` array.

`String Resources` are stored in a two dimensional array named `_ASYM_StringArray` as a user property of your book.

Typically the `ASYM_AddString()` function would be used at `Author` time to build up entries in the `String Resource` array. Seldom would you find a need to call this function at `Runtime`.

RETURNS

The function returns `true` if the string was successfully inserted into the `String Resource` array. Otherwise it returns `false`.

NOTES

Writing a ToolBook application which can be easily localized to other languages is a challenge for any programmer. To assist you in this effort, ToolBook provides a few functions [`ASYM_GetString()`, `ASYM_AddString()`, `ASYM_ClearString()`] to assist you in coding your OpenScript in such a way that other languages can pretty easily be swapped in. For all intents and purposes, any text you would typically hard code into your OpenScript logic should be wrapped in the `ASYM_GetString` function so that a localized string can instead be used, if present. The Examples below should be of help to further understand this concept.

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<string name>`	The name of a string resource.
`<string text>`	An expression that results in a string.

EXAMPLES

```
-- Inject a new String Resource via the command window
get ASYM_AddString("ConfirmExitCaption","Exiting Project...")

-- Inject a field full of English strings as String Resources
-- so that my German application contains English String too
-- Sample of text of field:
--   exitDlgCaption,Exit Project...
--   exitDlgText,Do you really want to Exit?
to handle buttonClick
    x = text of field "new strings"
    step k from 1 to textlineCount(x)
        curTextLine = textline k of x
        stringName = item 1 of curTextLine
        stringVal = item 2 of curTextLine
        get ASYM_AddString(stringName,stringVal)
    end
end
```

ASYM_ArrayPropertyDataType()

TB89R.SBK Array Function

SYNTAX `ASYM_ArrayPropertyDataType(<object>,<property>)`

DESCRIPTION Returns the data type of a user property array.

RETURNS The data type for the user property array represented as an integer.

The table below shows the correspondence between the return value and the data type.

0	Unknown	8	Custom value	16	Float
1	Logical	9	Enumerated list	17	Double
2	String	10	RGB color	18	Script
3	Word	11	HLS color	19	OCX (general)
4	Stack	12	Resource	20	OCX font
5	Rect	13	Indents	21	OCX picture
6	Point	14	Short	22	OCX color
7	DWORD	15	Long		

An untyped array will return `null`.

A value of `-1` will be returned if an error occurs during this operation. This could occur if the user property does not exist, or the property does not contain an array, or the object reference is not valid.

NOTES In order to utilize an array which is stored as a user property of an object, you must first move the array into an array defined within an OpenScript handler. However to do this you must first know what the data type of the array is. This function will return to you the data type of the array.

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<object>`	An object reference.
`<property>`	The name of a user property containing an array value for the object.

EXAMPLES

```
get ASYM_ArrayPropertyDataType(this book,"LastNames")
```

ASYM_Ask()

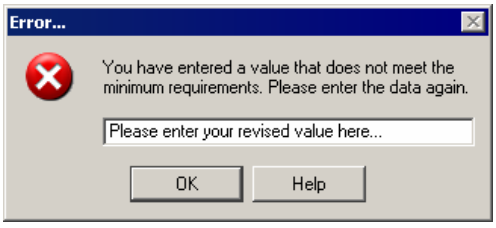
TB89R.SBK Dialog Box

SYNTAX

```
ASYM_Ask(<caption>,<text>,<icon>,<buttons>,<defText>,<width>,
         <position>,<frame>,<checkbox>,<radio>,<help>)
```

DESCRIPTION Displays a modal dialog which prompts the user to enter a response into an entry field.

This function provides a more flexible alternative to using the `Ask` command.



RETURNS A value of NULL is returned if the user does one of the following: press ESC, close the dialog using the X on the caption bar, or press the designated Cancel button.

Note: The ESC key will not function, and the X on the caption bar will not function, if you have not designated a Cancel button as described in the <buttons> parameter description.

Otherwise, the following 4 textlines are returned, separated by CRLFs:

```
<button>
<text>
<radio button>
<checkbox buttons>
```

RETURN VALUE	DESCRIPTION
<button>	A number representing which button was pressed. This number corresponds to the matching textline number passed in the <buttons> parameter.
<text>	A textline containing the text of the entry field that will appear in the Ask version of this dialog.
<radio button>	A number representing which radio button was checked. This number corresponds to the matching textline number passed in the <radio> parameter. This line will be included even if this feature was not utilized or even if no radio button was checked, which would result in an empty line.
<checkbox buttons>	A number representing which checkbox buttons were checked. This commas separated list of numbers corresponds to the matching textline numbers passed in the <checkbox> parameter. This line will be included even if this feature was not utilized or even if no checkbox buttons were checked, which would result in an empty line.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<caption>	The text to be applied as the caption of the dialog box. Maximum character length is 78. If null, no caption will appear.
<text>	Text to display in the body of the dialog box. RichText is permitted. If null, no text will appear.
<icon>	Valid values are: stop, question, exclamation, information You may instead pass a reference to an icon resource, such as icon "Apple" of this book If null no icon will be displayed in the dialog box.

PARAMETER	DESCRIPTION
<buttons>	There are two different methods for defining which buttons captions to display: STANDARD: Valid values are: OK, OKCancel, RetryCancel, AbortRetryIgnore, YesNo, YesNoCancel CUSTOM: A list of textlines separated by CRLF characters that contain up to ten button captions. These buttons will appear in a single row as command buttons just above the base of the dialog. If null, a single OK button will be used. The following special characters (use one or more) can be defined at the beginning of each caption, and will not appear in the actual caption text: + Indicates that this button will be the Default button. ? Indicates that this button will be the Help button. If used, pressing the button or pressing F1 will open the specified help file. x Indicates that this button will be used as the Cancel button. If used, ESC and the X in your caption bar will trigger this cancel button. (Indicates that this button will be disabled when the dialog is shown. / Indicates that all subsequent characters are part of the button's caption. This allows you to use one of the above characters as the first real character of the caption. [example: if your caption needs to read +5%, to protect the + character from being interpreted as a special character, write it as /+5%]
<defText>	Default text to be displayed in a single line editable text field. This will be positioned directly below <text>. RichText is not permitted.
<width>	The width (in pixels) of the dialog. If null the width of the dialog will be determined automatically. This value may be overridden if found to be too small (less than 200 pixels) or if less than the width of <buttons>, <radio> or <checkbox>.
<position>	The position (in pixels) of the upper left corner of the dialog. If null the position of the dialog will be centered on your screen.
<frame>	If you would like an 3D style inset frame to be shown surrounding the elements of the dialog, specify true, otherwise specify false. If null the frame will not be shown.
<radio>	A CRLF separated list of textlines containing up to 10 radio button captions. These buttons will appear above <buttons> stacked vertically and left justified with <text>. The radio buttons will behave in a mutually exclusive manner. If null, no radio buttons will appear. The following special characters (use one or more) can be defined at the beginning of each caption, and will not appear in the actual caption text: + Indicates that this button will be the Default button. ? Indicates that this button will be the Help button. If used, pressing the button or pressing F1 will open the specified help file. x Indicates that this button will be used as the Cancel button. If used, ESC and the X in your caption bar will trigger this cancel button. (Indicates that this button will be disabled when the dialog is shown. / Indicates that all subsequent characters are part of the button's caption. This allows you to use one of the above characters as the first real character of the caption. [example: if your caption needs to read +5%, to protect the + character from being interpreted as a special character, write it as /+5%]

PARAMETER	DESCRIPTION
<checkbox>	A CRLF separated list of textlines containing up to 10 checkbox button captions. These buttons will appear above <buttons>, and below <radio>, stacked vertically and left justified with <text>. If null, no checkbox buttons will appear. The following special characters (use one or more) can be defined at the beginning of each caption, and will not appear in the actual caption text: + Indicates that this button will be the Default button. ? Indicates that this button will be the Help button. If used, pressing the button or pressing F1 will open the specified help file. x Indicates that this button will be used as the Cancel button. If used, ESC and the X in your caption bar will trigger this cancel button. (Indicates that this button will be disabled when the dialog is shown. / Indicates that all subsequent characters are part of the button's caption. This allows you to use one of the above characters as the first real character of the caption. [example: if your caption needs to read +5%, to protect the + character from being interpreted as a special character, write it as /+5%]
<help>	Allows one of the <buttons> to open a help file by either clicking the button or pressing F1. This parameter is a comma separated list that can contain up to two items. Item 1 The name of the WinHelp file to use. If null, a file name is constructed by getting the current book's file name and changing the extension to .HLP. Otherwise, the file name must contain a path, or the file must be in the DOS path. Item 2 Optional. The Help Topic or Help Topic ID to look up. If null, the Contents tab will be opened. If null, no button will function as a Help button.

EXAMPLES

```
example_caption  = "Error..."
example_text     = "You have entered a value that does not " & \
                   "meet the minimum requirements."
example_icon     = icon "bad data" of this book
example_buttons  = "OK" & CRLF & "?Help"
example_defText  = "Please enter your revised value here..."
example_width    = null
example_position = null
example_frame    = false
example_check    = null
example_radio    = null
example_help     = "c:\tutor\tutor.hlp,447"
reply = ASYM_Ask(example_caption,example_text,example_icon,     \
                 example_buttons,example_defText,example_width, \
                 example_position,example_frame,example_check,  \
                 example_radio,example_help)
```

ASYM_AuthorResetPrompt

User Property

DESCRIPTION A book property that specifies if a dialog box is displayed with an option to reset the book when it is saved at Author level.

VALUES True or false.

If true, ASYM_Reset is sent to the book, which results in the message being sent to every page of the book.

ASYM_AutoBookmarks

User Property

DESCRIPTION A book property that specifies if bookmarks are written to a file when the book closes and reloaded when the book is reopened.

NOTES The bookmark file is created in the directory in which Windows stores .INI files. Bookmarks are read back only if the time stamps for the book and the bookmark file match.

The bookmark file contains a list of the pages with the `ASYM_BeenHere` and `ASYM_Done` properties set to `true` when the book is exited and the `leavePage` message is sent. When the system reads the file, it sets `ASYM_BeenHere` and `ASYM_Done` of each of the pages listed in the file to `true`. The system clears `ASYM_BeenHere` and `ASYM_Done` when the book is reset as a result of the `ASYM_Reset` message being sent.

VALUES `True` or `false`.

The default is `true`.

If `true`, the bookmarks are written to a file.

ASYM_AutoGlossary

User Property

DESCRIPTION A book property that specifies if a `glossary` entry is automatically displayed when the user clicks a hotword.

NOTES The hotword script calls `ASYM_PopGlossary()` to display the glossary entry.

VALUES `True` or `false`.

If true, the `glossary` entry is displayed when the hotword is clicked.

ASYM_AutoHotwords

User Property

DESCRIPTION A book property that specifies if any word in a field is automatically treated as a `hotword` that displays a glossary entry when clicked.

VALUES `True` or `false`.

If true, any word in the field is treated as a hotword that displays a glossary entry when clicked. The hotword script calls `ASYM_PopGlossary()` to display the glossary entry.

ASYM_AutoRemoveExt

User Property

DESCRIPTION A book property that controls whether or not unused extensions will be removed from the book when it is saved in the authoring environment.

NOTES Although technically Bound System Books are reported by `queryAddedExtension()` as extensions of a book, System Books don't have a reference count so therefore cannot be auto-removed. Only standard extensions such as ActiveX controls will be removed.

VALUES `True` or `false`.

If `true`, any extensions in the book that have a reference count of `0` will be removed when the book is saved in the authoring environment.

ASYM_BeenHere

User Property

DESCRIPTION A page property that specifies if a page has been visited by a user.

NOTES This property is used by `Bookmark` objects to indicate if a page referenced by the object has been displayed. This property is cleared automatically when `ASYM_Reset` is sent to the `page` or to the `book`.

`ASYM_BeenHere` is set to `true` automatically by a `leavePage` handler in the system book. This property is saved automatically to a `bookmark` file and restored when the book is restarted if `ASYM_AutoBookmarks` of the book is true.

VALUES `True` or `false`.

If `true`, the page has been visited.

ASYM_BookSysBooks

User Property

DESCRIPTION A book property that specifies the system books required by the book.

NOTES TB89R.SBK cannot be included in the `ASYM_BookSysBooks` list because it is required to interpret this property.

VALUES A string that contains a list of one or more system book names.

If necessary, the runtime system book (TB89R.SBK) attempts to find those system books, then push them onto the current system book list when the `enterBook` message is sent.

The system does not unlink these system books when the `leaveBook` message is sent.

ASYM_CheckObjectHyperlinks()

TB89R.SBK Hyperlink Functions

SYNTAX `ASYM_CheckObjectHyperlinks(<object>)`

DESCRIPTION Checks if the hyperlinks associated with an object point to valid destination pages, and returns a list of invalid hyperlinks.

RETURNS A list of the names of invalid hyperlinks, or `null` if no invalid hyperlinks were found.

NOTES This functionality is only available while Authoring your application, and is not a part of the runtime environment.

Although the function exists in the TB89R.SBK, it in turn calls another function in the TB89.SBK to do the actual work. This TB89.SBK book is an Authoring system book.

PARAMETERS

PARAMETER	DESCRIPTION
`<object>`	A valid object reference.

EXAMPLES

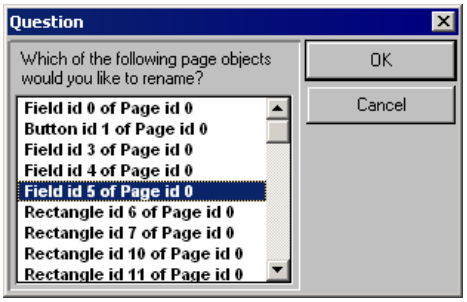
```
if ASYM_CheckObjectHyperlinks(button "index") = null
    request "Links are fine"
end
```

ASYM_ChooseFromTextLinesDlg()

TB89R.SBK Dialog Box

SYNTAX `ASYM_ChooseFromTextLinesDlg(<caption>,<prompt>,<textlines>,<default>)`

DESCRIPTION Shows a generic dialog box with a list box from which the user can select items. The dialog is able to auto-size itself to `1` of `3` possible widths, depending on the length of the textlines you are trying to display.



RETURNS A comma-separated list of two items.

The first item is either `true` (`OK`) or `false` (`Cancel`), depending on whether the user clicked `OK` or `Cancel` in the dialog box.

The second item is valid only if the first item is `true`, and is the `text` of the selected textline.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<caption>`	The caption for the dialog box window.
`<prompt>`	The prompt string to show above the list box.
`<textlines>`	`CRLF`-separated textlines to show in the list box.
`<default>`	The text of the line selected by default (can be `null`).

EXAMPLES

```
cap = "Question"
ppt = "Which of the following page objects would you like to rename?"
txt = listToTextline(objects of this page)
answ = ASYM_ChooseFromTextLinesDlg(cap,ppt,txt)
```

ASYM_ClearClipboard()

TB89R.SBK Clipboard Function

SYNTAX `ASYM_ClearClipboard()`

DESCRIPTION Clears the contents of the clipboard.

RETURNS `True` if successful.

`False` if the function is unable to open the clipboard; for example if another application is using it.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS No Parameters

EXAMPLES `get ASYM_ClearClipboard()`

ASYM_ClearHyperlink()

TB89HYP.SBK Hyperlink Functions

SYNTAX `ASYM_ClearHyperlink(<object>,<link name>)`

DESCRIPTION Clears a hyperlink from the list of hyperlinks associated with an object.

RETURNS `True` if the hyperlink was successfully removed from the object.

`False` if it was not.

If `false`, check `sysError` for the cause of error.

NOTES This function is provided by the TB89HYP.SBK. To successfully use this function in Runtime, you must first add the TB89HYP.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89HYP.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89HYP.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<object>`	A valid reference to an object.
`<link name>`	The name of the hyperlink to serve as a trigger, or the standard trigger `buttonClick`.

EXAMPLES

```
-- clear the custom hyperlink
get ASYM_ClearHyperlink(Button "index","adminLink")

-- clear the standard hyperlink
get ASYM_ClearHyperlink(Button "index","buttonClick")
```

ASYM_ClearString()

TB89R.SBK String Functions

SYNTAX `ASYM_ClearString(<string name>)`

DESCRIPTION Removes a `String Resource` entry in the `String Resource` array.

`String Resources` are stored in a two dimensional array named `_ASYM_StringArray` as a user property of your book.

Typically the `ASYM_ClearString()` function would be used at `Author` time to build up entries in the `String Resource` array. Seldom would you find a need to call this function at `Runtime`.

RETURNS The function returns `true` if the string was successfully inserted into the `String Resource` array. Otherwise it returns `false`.

NOTES Writing a ToolBook application which can be easily localized to other languages is a challenge for any programmer. To assist you in this effort, ToolBook provides a few functions [`ASYM_GetString()`, `ASYM_AddString()`, `ASYM_ClearString()`] to assist you in coding your OpenScript in such a way that other languages can pretty easily be swapped in. For all intents and purposes, any text you would typically hard code into your OpenScript logic should be wrapped in the `ASYM_GetString` function so that a localized string can instead be used, if present. The Examples below should be of help to further understand this concept.

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<string name>`	The name of a string resource.

EXAMPLES

```
-- Remove an old String Resource via the command window
get ASYM_ClearString("ConfirmExitCaption")

-- Clear a field full of String Resources
to handle buttonClick
    x = text of field "old strings"
    step k from 1 to textlineCount(x)
        curTextLine = textline k of x
        get ASYM_ClearString(curTextLine)
    end
end
```

ASYM_CompareByCase()

TB89R.SBK String Functions

SYNTAX `ASYM_CompareByCase(<string1>,<string2>)`

DESCRIPTION This function compares two strings to see if they are identical.

This function is `case sensitive` whereas all other string operators and functions in ToolBook are `case insensitive`.

RETURNS Returns `true` if the two strings are identical. `False` is returned if they are not identical.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<string1>`	An expression that results in a string value.
`<string2>`	An expression that results in a string value.

EXAMPLES

```
-- Case sensitive comparison
to handle buttonClick
    x = text of field "Password"
    if ASYM_CompareByCase(x,"TableTOP") = true
        -- password is correct
        go to page "Begin"
    end
end

-- Case insensitive comparison
to handle buttonClick
    x = text of field "Password"
    if x = "TableTOP"
        -- password is correct
        -- but user could have typed tAbLeToP for all we know
        go to page "Begin"
    end
end
```

ASYM_CurrentDirectory()

TB89R.SBK File Function (Directory)

SYNTAX `ASYM_CurrentDirectory()`

DESCRIPTION Gets the current directory.

RETURNS A string that contains the current drive and directory, ending with a backslash (\).

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS No Parameters

EXAMPLES
```
curDir = ASYM_CurrentDirectory()
```

ASYM_DoHyperlink()

TB89HYP.SBK Hyperlink Functions

SYNTAX
```
ASYM_DoHyperlink(<object>,<trigger>)
```

DESCRIPTION Executes a Hyperlink. Navigates to the page specified for the hyperlink, or displays the page in a popup window with an optional transition effect specified for the hyperlink.

RETURNS If successful, the function returns `true`; otherwise, it returns `false`.

NOTES This function is provided by the TB89HYP.SBK. To successfully use this function in Runtime, you must first add the TB89HYP.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89HYP.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89HYP.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<object>`	A valid reference to the object that owns the hyperlink to execute.
`<trigger>`	The name of the trigger event of the hyperlink to execute. The system automatically executes hyperlinks with the trigger event named `buttonClick` when a user clicks on the object. For other triggers, you must call this function, passing the appropriate trigger event name.

EXAMPLES
```
-- trigger a custom hyperlink for this object
get ASYM_DoHyperlink(button "extra","customLink")

-- trigger the standard hyperlink for this object
get ASYM_DoHyperlink(button "extra","buttonClick")
```

ASYM_Done

User Property

DESCRIPTION A page property that you can set in a script for any purpose specific to your application, such as keeping track of completed questions, or determining whether you have already played an animation on a page.

NOTES The value of `ASYM_Done` is saved automatically to a `bookmark` file and restored when the book is restarted if `ASYM_AutoBookmarks` of the book is `true`. This property is not set automatically. You can set it or clear it at will. `ASYM_Done` is cleared automatically when `ASYM_Reset` is sent to the page or to the book.

VALUES `True` or `false`.

ASYM_Draggable

User Property

DESCRIPTION A property of graphic and draw objects that specifies if an object can be dragged at `Reader` level.

NOTES This property is similar to the `defaultAllowDrag` property, but when the object is dragged, the object itself moves, rather than a cursor or graphic set for the `dragImage` property.

This property is used by some question objects to create drag behavior.

VALUES `True` or `false`.

If `true`, the object can be dragged.

ASYM_Ellipsize()

SYNTAX `ASYM_Ellipsize(<string>,<length>,<side>)`

DESCRIPTION Replaces the excess part of a string with an ellipsis. If the string contains a backslash (\), it is assumed to be a file name and the function attempts to return a readable name; in this case the result may be shorter than the specified length.

Note that you may want to use the `getEllipsisByCharCount32()` instead. It is a newer function which incorporates more flexibility and is much faster to execute.

RETURNS A string of text with the added ellipsis.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<string>`	An expression that results in a string value.
`<length>`	The maximum character length (including the ellipsis) of the returned string. If the string is shorter or equal to that length, no ellipsis is added.
`<side>`	A value of `left`, `right`, or `center` to specify the side of the string to which an ellipsis is added. You must use quotation marks for this parameter value.

EXAMPLES

```
to handle buttonClick
    x = text of field "Story3"
    shortX = ASYM_Ellipsize(x,50,"left")
    request "Of all your stories, the one I liked best was " & \
            "the one that started out: " & shortX
end
```

ASYM_EllipsizeFileToField()

SYNTAX `ASYM_EllipsizeFileToField(<fieldRef>,<filename>[,<fileOption> [,<dirOption>]])`

DESCRIPTION Ellipsizes a filename to fit within the width of a field.

RETURNS The ellipsized version of the filename, based on the width of the field and it's current font settings.

If an error occurs, the function returns `null`.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<fieldRef>`	An object reference to a field or recordfield.
`<filename>`	The filename to be ellipsized.
`<fileOption>`	An optional parameter to control how the filename is ellipsized: `rue` The entire filename will be returned. This is the default setting. `alse` Implies the filename may be ellipsized, if it exceeds the maximum width.
`<dirOption>`	An optional parameter to control how directory names are ellipsized: `true` Shortens the path by removing the entire directory name at once, until the total length is less than the maximum. This is the default setting. `false` Shortens the path by removing individual characters, until the total length is less than the maximum.

EXAMPLES

```
to handle buttonClick
    x = text of field "Story3"
    shortX = ASYM_Ellipsize(x,50,"left")
    request "Of all your stories, the one I liked best was " & \
            "the one that started out: " & shortX
end
```

ASYM_ExpandString()

TB89R.SBK String Functions

SYNTAX

```
ASYM_ExpandString(<string image>,<param>[,<param>])
```

DESCRIPTION

Replaces placeholders in a string with parameters.

This function works identically to the function: `expandString()`

RETURNS

Returns the modified string.

NOTES

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<string image>`	A string that can contain placeholders to be replaced by parameters. Placeholders are indicated in the form `%<n>`, where `<n>` corresponds to the parameter number. For example `%1` represents the first parameter, `%2` represents the second parameter, and so on. If by chance you want to include the `%` character in your original string, and not have it represent a replacement parameter, use two `%` characters instead of one. Example: `ASYM_ExpandString("Buy 2%% milk at: %1","market")` The same placeholder can appear more than once in the `<string image>`. If the parameter referenced by a numbered placeholder does not exist, the placeholder is replaced by a `null` string.
`<param>`	A string or expression that evaluates to a string. You can pass up to `99` parameters to this function.

EXAMPLES

```
-- Displays the message: David, you earned 11 points. Great job David!
pts = 11
uName = "David"
msg = "%1, you earned %2 points. Great job %1!"
request ASYM_ExpandString(msg,uName,pts)
```

ASYM_FileToPrinter()

TB89R.SBK Printing Function

SYNTAX

```
ASYM_FileToPrinter(<file name>,<options>,<window handle>,<bShowDlg>)
```

DESCRIPTION

Prints the contents of a raw text file to the current default printer.

This function automatically wraps the text to fit on the printed page, with fixed (8-space) tabs. Only one style of formatting is available for each document.

RETURNS

If successful, the function returns `true`.

Otherwise, it returns `false`.

NOTES

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<file name>	The name of a file.
<options>	A list of textlines (CRLF-separated) that specify the following options: Document Name — A name for the document to be printed. Font Face — A valid font name. Font Style list — Bold, italic, underline, or strikeout. Font Size — A valid font size. Margins in inches — A string that contains a list of four numbers that correspond to the <left>, <top>, <right>, and <bottom> margins.
<window handle>	The window handle of the parent window of the Cancel dialog box.
<bShowDlg>	1 Shows the Cancel dialog box, 0 Does not show the Cancel dialog box.

EXAMPLES

```
to handle buttonClick
    fileN = "c:\master.log"
    docName = "Test"
    fontF = "Times New Roman"
    fontS = "Bold"
    fontSz = 10
    marg = "1.5,1,1.5.1"
    opts = docName & CRLF & fontF & CRLF & fontS & \
           CRLF & fontSz & CRLF & marg
    hndl = windowHandle of mainWindow
    showDlg = 1
    get ASYM_fileToPrinter(fileN,opts,hndl,showDlg)
end
```

ASYM_FindExecutableFile()

TB89R.SBK File Function (General)

SYNTAX

```
ASYM_FindExecutableFile(<filespec>)
```

DESCRIPTION

Gets the complete file name of the executable file associated with <filespec>.

RETURNS

The full file name [in short filename form] of an executable file, including the path.

Returns null if no executable file is associated in Windows with the specified file or extension.

NOTES

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<filespec>	Either the name of a file, or a file extension mask in the format *.<extension>

EXAMPLES

```
-- See if user has a program to open DOC files like Word or Wordpad
if ASYM_FindExecutableFile("*.doc") = null
    request "Sorry, can't open the form because you don't have " & \
            "the needed software installed to open it."
end
```

ASYM_FindHyperPage()

TB89HYP.SBK Hyperlink Functions

SYNTAX `ASYM_FindHyperPage(<page>,<alt pgref>,<book>)`

DESCRIPTION Locates the specified page, if it exists. Use this function to locate a page before navigating to it, or to convert an ambiguous page reference into a formal page reference.

RETURNS A valid page reference, or `null` if the destination page could not be located.

NOTES This function is provided by the TB89HYP.SBK. To successfully use this function in Runtime, you must first add the TB89HYP.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89HYP.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89HYP.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<page>`	The `name` of a page, or the `ID` number of the page in the form `ID <number>`.
`<alt pgref>`	The `ID` number of the page in the form `ID <number>`. This parameter is used if the page cannot be found by its name (for example, if you renamed the page after creating it).
`<book>`	The file name of a book. If `null`, the current book is assumed. If the `<page>` parameter includes a book reference in addition to the page, this parameter is ignored. When only the file name is specified (no path), `ASYM_HyperPath` is searched to locate the destination book.

EXAMPLES

```
pgRef = ASYM_FindHyperPage("index","id 44",null)
if pgRef <> null
    go page pgRef
else
    request "Sorry, Index page not found."
end
```

ASYM_FindPathFile()

TB89R.SBK File Function (General)

SYNTAX `ASYM_FindPathFile(<file name>,<path>)`

DESCRIPTION This function will attempt to locate a specified file in a variety of specified paths.

RETURNS If the file could be found in one of the paths that you specified, the entire path of the file is returned.

Otherwise the function returns `null`.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	A file name (does not include the path).
`<path>`	A list of one or more path strings. If `<path>` is `null`, the current `DOS` path setting is used.

EXAMPLES

```
-- when linking a DLL
dllName = "super.dll"
utilDLL = ASYM_FindPathFile(dllName,null)
if utilDLL <> null
    linkDLL utilDLL
    ...
    end
else
    request "Cannot find" && dllName && "."
    send exit
end

-- when locating an external file
paths = "c:\,c:\project\documents\,c:\project\backupfiles\"
fName = "scores.txt"
loc = ASYM_FindPathFile(fName,paths)
if loc = null
    request "File cannot be located"
else
    run loc
end
```

ASYM_FirstNavigablePage()

TB89R.SBK Navigation

SYNTAX `ASYM_FirstNavigablePage()`

DESCRIPTION Gets the page reference of the first page of the current book with the `skipNavigation` property set to `false`.

RETURNS A page reference of the first navigable page.

If no navigable page is found or if the first navigable page is the current page, the function returns `null`.

NOTES This function steps through the book's pages, starting with the first page, until it finds a valid page. To avoid performance degradation, use this function sparingly in large books that contain many pages with `skipNavigation` set to `true`.

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS No parameters.

EXAMPLES

```
-- Handler in script of button "First" of some background
notifyBefore enterPage
    my enabled = (ASYM_FirstNavigablePage() <> null)
end
```

ASYM_FreeDiskSpace()

TB89R.SBK File Function (General)

SYNTAX `getFreeDiskSpace(<disk drive>)`

DESCRIPTION Gets the number of free bytes on the specified disk drive.

RETURNS If no error occurs, `getFreeDiskSpace32()` returns the number of free bytes on the specified disk.

Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<disk drive>	The letter designating the disk drive for which you want the number of free bytes.

EXAMPLES

```
freeSp = ASYM_FreeDiskSpace("c")
```

ASYM_FullScreen

User Property

DESCRIPTION A book property that specifies if the book is displayed in the entire screen at `Reader` level.

NOTES This has the effect of maximizing the application window to fill the monitor. It will also cover the Windows taskbar if shown.

VALUES `True` or `false`.

If `true`, the book is displayed at full screen at `Reader` level.

ASYM_GetFileDate()

TB89R.SBK File Function (Attribute)

SYNTAX

```
ASYM_GetFileDate(<filespec>,[<format string>])
```

DESCRIPTION Returns the date and time the specified file was last updated.

This function is similar to `getFileDate32()`, but it handles localization and dates beyond the year `2000` more efficiently, because you can specify the format of the return value.

RETURNS If no error occurs, this function returns a string containing the date and time, formatted according to the `<format string>` parameter.

If an error occurs, the function returns `NULL` and `sysError` is set to one of these values:

```
 -2  Specified file was not found.
 -3  Specified path was invalid.
-20  Memory allocation error.
```

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<filename>	A fully-qualified path to the file to be examined.
<format string>	A valid ToolBook date and time format string, as defined for the `Format` command, which specifies the format of the returned date and time. If this parameter is not specified, the returned date and time will be formatted according to the values of `sysDateFormat` and `sysTimeFormat`. If `sysDateFormat` includes time information, then it alone will determine the format of the return value, otherwise a combination of `sysDateFormat` and `sysTimeFormat` is used to format the return value.

EXAMPLES

```
-- Get the date when a book was last changed, in sysDateFormat
bookDate = ASYM_GetFileDate(name of this book,sysDateFormat)

-- Get the date a file was last changed, with a four digit year
fileDate = ASYM_GetFileDate(fileName,"mm/dd/yyyy h24:min:sec")
```

ASYM_GetFileVersion()

TB89R.SBK File Function (Attribute)

SYNTAX `ASYM_GetFileVersion(<filename>)`

DESCRIPTION Reads a file's header and returns the file version information.

RETURNS A list of four decimal numbers.

The first two numbers represent the most significant and least significant portions of the major version number. The last two numbers represent the most significant and least significant portions of the minor version number.

If the file does not exist or has no version information, the function returns `NULL`.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<filename>`	A fully-qualified path to the file to be examined.

EXAMPLES

```
get ASYM_GetFileVersion("c:\tlane.exe")
```

ASYM_GetHyperlinks()

TB89HYP.SBK Hyperlink Functions

SYNTAX `ASYM_GetHyperlinks(<object>)`

DESCRIPTION Gets a list of hyperlinks associated with an object.

RETURNS A list of hyperlinks in the form of textlines.

Each text line contains information about one hyperlink in a comma-separated list, and includes:

ITEM	DESCRIPTION
1	The name of the hyperlink. Example: `buttonClick`
2	The name of the destination page (or "`ID <number>`"), or a generic link tag such as `<next>`, `<previous>`, `<first>`, or `<last>`, or a `URL` reference.
3	The alternative name or `idNumber` of the destination page, if the second item is not a generic link tag.
4	The name of the destination book (if different from the book that owns the object).
5	The popup window style: `shadow`, `shadowAutoClose`, `thickFrame`, `thinFrame`, or `dialogFrame`.
6	The link type: `jump` or `popup`.
7	The `sysCursor` number.
8	`True` if this is a `URL` hyperlink, otherwise `false`.
9	The value of the `In Window` setting: `<targetWindow>`, `<mainWindow>`, or `viewer id 0`.

NOTES This function is provided by the TB89HYP.SBK. To successfully use this function in Runtime, you must first add the TB89HYP.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89HYP.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89HYP.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<object>	A valid object reference.

EXAMPLES

```
-- See if this object has a URL hyperlink, and if so, display it.
hyps = textline 1 of ASYM_GetHyperlinks(button "link")
if item 8 of hyps = true
    request item 2 of hyps
end
```

ASYM_GetProductVersion()

TB89R.SBK File Function (Attribute)

SYNTAX

```
ASYM_GetProductVersion(<filename>)
```

DESCRIPTION

Reads a file's header information and returns the product version information.

RETURNS

A list of four decimal numbers.

The first two numbers represent the most significant and least significant portions of the product major version number. The last two numbers represent the most significant and least significant portions of the product minor version number.

If the file does not exist or has no version information, the function returns `NULL`.

NOTES

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<filename>	A fully-qualified path to the file to be examined.

EXAMPLES

```
get ASYM_GetFileVersion("c:\tlane.exe")
```

ASYM_GetString()

TB89R.SBK String Functions

SYNTAX

```
ASYM_GetString(<string name>[,<default>)]
```

DESCRIPTION

Looks up an entry in the `String Resource` array.

`String Resources` are stored in a two dimensional array named `_ASYM_StringArray` as a user property of your book.

RETURNS

This function will return the string entry stored in the `String Resource` array for the particular `<string name>` specified.

If the string was not found but a `<default>` was specified, the function returns the `<default>` string.

If no string is found and no `<default>` is specified, the function returns the value of `<string name>`.

NOTES

Writing a ToolBook application which can be easily localized to other languages is a challenge for any programmer. To assist you in this effort, ToolBook provides a few functions [`ASYM_GetString()`, `ASYM_AddString()`, `ASYM_ClearString()`] to assist you in coding your OpenScript in such a way that other languages can pretty easily be swapped in. For all intents and purposes, any text you would typically hard code into your OpenScript logic should be wrapped in the `ASYM_GetString` function so that a localized string can instead be used, if present. The Examples below should be of help to further understand this concept.

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<string name>	The name of a string resource.
<default>	An optional default value to be returned if the string is not found.

EXAMPLES

```
-- Display an exit confirmation dialog
to handle buttonClick
    cap = ASYM_GetString("ConfirmExitCaption","Exit...")
    txt = "Are you sure you really want to exit now?"
    txt = ASYM_GetString("ConfirmExitBodyText",txt)
    answ = ASYM_Request(cap,txt,"stop","YesNo")
    if textline 1 of answ = 1
        send exit
    end
end

if sysTime > maxAllowableTime
    newCap = ASYM_GetString("lateSubmission","Submit Late Entry")
    caption of button "exit" = newCap
end
```

ASYM_GetSystemVar()

TB89R.SBK Variable Functions

SYNTAX `ASYM_GetSystemVar(<variable name>)`

DESCRIPTION Gets the value of a system variable in the current ToolBook instance.

RETURNS The value of the system variable.

If the system variable does not exist, the function returns `null`.

NOTES Due to the way OpenScript optimizes compilation, this function may not return a valid result when called in the same handler where the system variable was declared.

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very p ossible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<variable name>	The name of a system variable.

EXAMPLES

```
request "Your total score was:" && ASYM_GetSystemVar("_score")
```

ASYM_GetTempFile()

TB89R.SBK File Function (Temp)

SYNTAX `ASYM_GetTempFile(<prefix>)`

DESCRIPTION Generates a temporary file in the TEMP directory.

Note that this function actually creates the temporary file. The temporary file will not be automatically removed. it is the responsibility of the developer to clean up temporary files.

RETURNS The filename and path [in short file name form] of the created temporary file.

`NULL` if the function was unable to create the file.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<prefix>	A string to be used as the file's prefix. Only the first 3 characters of the prefix will be used. If no prefix is specified, then TMP will be used.

EXAMPLES

```
-- Generate a Temp file with a LOG prefix:
-- Produces: C:\DOCUME~1\DENNYD~1\LOCALS~1\Temp\log18.tmp
x = ASYM_GetTempFile("log")
```

ASYM_GlossaryName

User Property

DESCRIPTION A book property that specifies the name of the `glossary` file if you want to use a book different from GLOSSARY.TBK.

VALUES A string that contains the name of the book to be used as a `glossary` file.

ASYM_GlossaryPage()

TB89R.SBK Glossary Function

SYNTAX

```
ASYM_GlossaryPage(<term>,<book>)
```

DESCRIPTION Gets a reference to a glossary page for the specified term.

RETURNS A page reference, or `null` if no such page could be located.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<term>	The name of the term to find (should be a page name).
<book>	An optional argument specifying the book in which to find the term. If `null`, the system searches for a page by the specified name in the following order: 1. background "`glossary`" of current book, if it exists. 2. The file "`GLOSSARY.TBK`", if it can be found in one of the directories specified by the ASYM_HyperPath property of the book. If the file name does not include a path, the system attempts to locate the book in the directories specified by `ASYM_HyperPath`.

EXAMPLES

```
to handle buttonClick
    ask "Verify existence of what Glossary Term?"
    if ASYM_GlossaryPage(IT,null) = null
        request "Entry does not exist."
    else
        request "Entry was found."
    end
end
```

ASYM_GoToPage()

TB89HYP.SBK Navigation

SYNTAX `ASYM_GoToPage(<page>,<transition>,<cursor>)`

DESCRIPTION Navigates to the specified page using the optional special effect and/or cursor.

Use this function instead of the `go` command to simplify scripts in your application. You can use `ASYM_FindPage()` to construct a guaranteed `<page>` at runtime.

RETURNS If successful, the function returns `true`; otherwise, it returns `false`.

NOTES This function is provided by the TB89HYP.SBK. To successfully use this function in Runtime, you must first add the TB89HYP.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89HYP.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89HYP.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<page>`	A complete page reference, which must include a book name if not in the current book. If the page reference is invalid, an `Execution Suspended` error may occur in a system book.
`<transition>`	This parameter is optional. A string of options that describe the transition effect used in navigating to the page. The value can be `null`. The effect string uses the following syntax: `<effect> <direction> <destination> <speed>` For a list of valid effect strings, see `transition`.
`<cursor>`	An optional `sysCursor` number, or `null`.

EXAMPLES

```
get ASYM_GoToPage(page "results","zoom in",4)
```

ASYM_HasHyperlinks()

TB89HYP.SBK Hyperlink Functions

SYNTAX `ASYM_HasHyperlinks(<object>)`

DESCRIPTION Checks if an object is currently configured with a hyperlink.

RETURNS `True` or `false`.

NOTES This function is provided by the TB89HYP.SBK. To successfully use this function in Runtime, you must first add the TB89HYP.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89HYP.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89HYP.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<object>`	A valid object reference.

EXAMPLES

```
-- Does rectangle X4 have a hyperlink configured
hasHyp = ASYM_HasHyperlinks(rectangle "x4")
```

ASYM_HyperPath

TB89HYP.SBK Hyperlink Property

SYNTAX `ASYM_HyperPath of book <book>`

DESCRIPTION A property that specifies the directories in which the system searches for hyperlinked books. The system uses the value of this property to determine where other books should be located when no path is specified for them.

VALUES A string that contains a list of the paths (can be relative) to be searched. This value can be `null` if the hyperlinked books are located in the same directory.

The directory of the current book is always searched first. If you install your application on another system and it contains links to other books in different directories, you may need to modify the `ASYM_HyperPath` property of the main application book.

Example: If the current book is `c:\books\mybook.tbk` and one of the hyperlinked books is `c:\books\resource\testitem.tbk`, then `ASYM_HyperPath` of the book can be set to `resource\`.

The book file path is always the first item of the value of `ASYM_HyperPath`; it is automatically pushed onto the value of the property when you set `ASYM_HyperPath` (unless it is already in the list). You can use the keyword `<BookDir>` as a placeholder for the book path.

NOTES This functionality is provided by the TB89HYP.SBK. To successfully use this it in Runtime, you must first add the TB89HYP.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89HYP.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89HYP.SBK will already be there.

Note that this is not a standard book property or even a user property. In fact, when you `get` or `set` the value of `ASYM_HyperPath`, a `to get ASYM_HyperPath` and `to set ASYM_HyperPath` set of handlers in the TB89HYP.SBK intercept and handle the tasks. If you set the value of `ASYM_HyperPath`, you will notice a user property will appear in your book named `_asym_HyperPath`. Never directly modify this `_asym_HyperPath` property, always use `ASYM_HyperPath` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<book>`	A valid reference to a book.

EXAMPLES

```
set ASYM_HyperPath of this book to null

get ASYM_HyperPath of this book

set ASYM_HyperPath of this book to "extra\glossary\"
```

ASYM_IniFile()

TB89R.SBK INI Functions

SYNTAX `ASYM_IniFile()`

DESCRIPTION Gets the name of the `.INI` file in which runtime and `Reader` level preferences and information are stored.

RETURNS The full path and filename of the Runtime `INI` file (`TB89R.INI`).

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS No Parameters

EXAMPLES

```
--Adds a custom entry in the runtime .INI file
get setIniVar("Corporate","DateAccessed",sysDate",ASYM_IniFile())
```

ASYM_IsDirectory()

TB89R.SBK File Function (Directory)

SYNTAX `ASYM_IsDirectory(<directory name>)`

DESCRIPTION Checks if the specified directory exists.

RETURNS `True` to indicate the directory exists.

`False` if it does not.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<directory name>`	A complete directory name, including the drive, such as `c:\mybooks`. The directory specification can end in a backslash (\).

EXAMPLES

```
if ASYM_IsDirectory("c:\project") = false
    request "It does not appear that you have installed module X33."
end
```

ASYM_IsDirectoryWriteable()

TB89R.SBK File Function (Directory)

SYNTAX `ASYM_IsDirectoryWriteable(<directory name>)`

DESCRIPTION Checks if a file can be created in a directory.

If the directory does not exist or the drive is not ready, an error message is displayed to indicate the problem.

RETURNS `True` to indicate a file can be created in the specified directory.

`False` if it cannot.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<directory name>`	A complete directory name, including the drive, such as `c:\mybooks`. The directory specification can end in a backslash (\).

EXAMPLES

```
if ASYM_IsDirectoryWriteable("c:\project") = false
    request "Sorry, can't write file to disk."
end
```

ASYM_IsDriveReady()

TB89R.SBK File Function (Drive)

SYNTAX `ASYM_IsDriveReady(<drive letter>)`

DESCRIPTION Checks if a drive exists and is ready for use.

For a removable media drive (floppy or CD), this function checks the status of the drive without triggering the usual Windows critical drive error message when the media is not properly inserted.

RETURNS `True` if the drive is ready.

`False` if the drive is not ready.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<drive letter>	The letter designating the disk drive.

EXAMPLES

```
dr = "A"
while not ASYM_IsDriveReady(dr)
    clear sysError
    str = "Please put a formatted floppy disk in drive" && dr & "."
    request str with "OK" or "Cancel"
    if sysError is cancel
        break
    end
end
```

ASYM_IsFile()

TB89R.SBK File Function (General)

SYNTAX

```
ASYM_IsFile(<file name>)
```

DESCRIPTION

Checks if the specified file.

This function uses the function `fileExists32()`, except this function returns `true` or `false`, while `fileExists32()` returns a number.

RETURNS

`True` if the file.

`False` if the file does not.

NOTES

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<file name>	A complete file name.

EXAMPLES

```
fName = "intro.txt"
if ASYM_IsFile(fName) = true
    openFile fName
    readFile fName to eof
    closeFile fName
    text of field "Introduction" = IT
end
```

ASYM_IsFileAvailable()

TB89R.SBK File Function (General)

SYNTAX

```
ASYM_IsFileAvailable(<file name>)
```

DESCRIPTION

Checks if the specified file exists and if it can be opened without a sharing violation.

This function does not show the usual Windows critical error dialog box when the file cannot be opened.

RETURNS

`True` if the file exists and can be opened without a file sharing violation.

`False` if the file does not exist or cannot be opened.

NOTES

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<file name>	A complete file name.

EXAMPLES

```
fName = "intro.txt"
if ASYM_IsFileAvailable(fName)
    openFile fName
    readFile fName to eof
    closeFile fName
    text of field "Introduction" = IT
end
```

ASYM_IsNumber()

TB89R.SBK Validation

SYNTAX `ASYM_IsNumber(<value>)`

DESCRIPTION Checks if the specified value is a number.

This function works specifically with integer and decimal numbers.

It returns false for numbers that include special characters, such as $4.00.

RETURNS `True` if the value is a number.

`False` if not.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<value>	A literal numeric value or the name of a variable.

EXAMPLES

```
val = text of field "answer"
if ASYM_IsNumber(val)
    x = val * 1.5
end
```

ASYM_IsScored

User Property

DESCRIPTION A book property that specifies if the book's score is evaluated and posted to a CMS when the `leaveApplication` message is sent.

NOTES This property also specifies if the score is written to the end of the session log, if logging is used.

The score is posted to the CMS only if the book is run from the CMS.

VALUES `True` or `false`.

If `true`, the score is evaluated and posted to the CMS when the `leaveApplication` message is sent.

ASYM_ItemInList()

TB89R.SBK String Functions

SYNTAX `ASYM_ItemInList(<item>,<list>)`

DESCRIPTION Determines if the specified `<item>` exists in a comma separate `<list>` of items.

This function is equivalent to saying: `(itemOffset(x,y) <> 0)`

RETURNS The function returns `true` if the item is in the list, and `false` if the item is not in the list.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<item>	An expression that results in a string value.
<list>	An expression that results in a list.

EXAMPLES

```
to handle buttonClick
    possibleAnswers = "apple,banana,orange,pineapple"
    ask "What is one of my favorite fruits?"
    answ = IT
    if ASYM_ItemInList(answ,possibleAnswers)
        request "You are correct."
    else
        request "Sorry, incorrect!"
    end
end
```

ASYM_ItemOffset()

TB89R.SBK String Functions

SYNTAX `ASYM_ItemOffset(<item>,<list>)`

DESCRIPTION Locates the first occurrence of an item in a list.

This function works identically to the `itemOffset()` function.

RETURNS Returns the item position in which `<item>` was found in `<list>`. If `<item>` cannot be found within `<list>` then `0` is returned.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<item>	An expression that results in a string value.
<list>	An expression that results in a list.

EXAMPLES

```
to handle buttonClick
    lst = "apple,banana,orange,pineapple"
    ask "What are you looking for?"
    answ = IT
    pos = ASYM_ItemOffset(answ,lst)
    if pos > 0
        request "Yes, that item is in position" && pos
    else
        request "Sorry, didn't find a match"
    end
end
```

ASYM_LastNavigablePage()

TB89R.SBK Navigation

SYNTAX `ASYM_LastNavigablePage()`

DESCRIPTION Gets the page reference of the last page of the current book with the `skipNavigation` property set to `false`.

RETURNS A page reference to the last navigable page. If no navigable page is found or if the last navigable page is the current page, the function returns `null`.

NOTES This function steps through the book's pages until it finds a valid page. To avoid performance degradation, use this function sparingly in large books that contain many pages with `skipNavigation` set to `true`.

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS No parameters.

EXAMPLES

```
notifyBefore enterPage
    my enabled = (ASYM_LastNavigablePage() <> null)
end
```

ASYM_LogAppend

User Property

DESCRIPTION A book property that specifies if the log file is appended with logs from subsequent sessions.

NOTES When you set this property in an `enterApplication` handler, always set it before a `forward` statement in the same handler to ensure that its value is recognized by the system when the log is started.

VALUES `True`, `false`, or `null`.

If `true` and the `ASYM_LogName` property of the book is not `null`, logs for subsequent sessions are appended to the existing log file.

If `false` or `null`, new log files replace any existing log file with the same name.

ASYM_LogDestination

User Property

DESCRIPTION A book property that specifies the default destination directory for log files.

NOTES When you set this property in an `enterApplication` handler, always set it before a `forward` statement in the same handler to ensure that its value is recognized by the system when the log is started.

VALUES A string that contains one of the following values: `<sameDir>`, `<iniDir>`, `<tempDir>`, `<floppy>`, `<ask>`, `<fax>`, `<email>`, `<printer>`, `<special>`, or `null`.

VALUES	DESCRIPTION
`<sameDir>`	The log is stored in the same directory as the book.
`<iniDir>`	The log is stored in the directory where Windows stores .INI files.
`<tempDir>`	The log is stored wherever Windows applications store temporary files.
`<floppy>`	The system attempts to locate a writable floppy drive with a disk. If more than one floppy drive qualifies, the system asks the user to specify the drive.
`<fax>` or `<email>`	The log is stored in a temporary file, and the log is deleted after being sent (if successful).
`<special>`	The book uses the value of the system variable `s_ASYM_LogDir` for the location of the log file. If the value is `null`, the log is stored in the directory where Windows stores .INI files.
`<printer>`	The log file is sent to the printer at the end of the session, then deleted.
`null`	The following directories are searched in this order: `<sameDir>`, `<iniDir>`, `<tempDir>`, and the user is asked to confirm or specify another directory.

EXAMPLES

```
ASYM_LogDestination = "<printer>"
```

ASYM_LogEncrypt

User Property

DESCRIPTION A book property that specifies if the session log is encrypted.

VALUES `True`, `false`, or `null`.

If `true`, and the `ASYM_LogEncryptKey` property of the book is not `null`, the contents of the log are encrypted.

If `false` or `null`, the log is not encrypted.

ASYM_LogEncryptKey

User Property

DESCRIPTION A book property that specifies the encryption key for logs.

NOTES The system creates this key by encrypting the password provided by the author in the Book Properties - Instructor Options dialog box. The encryption key cannot be used to decrypt the log file; only the original password is accepted.

VALUES A string that contains the encryption key.

ASYM_LogHeading

User Property

DESCRIPTION A book property that specifies the header for the beginning of the transcript written to a session log file.

VALUES A string that contains the text used as the header.

ASYM_LogName

User Property

DESCRIPTION A book property that specifies the name for the session log file that is created when the book first runs.

NOTES You must first set the value of `ASYM_LogType` to create a log.

VALUES A string that contains a log file name.

If `null`, a default name is assigned to the log file in the form `LOGxxxx.LOG`, where `xxxx` is a number incremented automatically if another file exists with the same name.

ASYM_LogOptions

User Property

DESCRIPTION A book property that specifies the settings for the book's logging options.

VALUES A string that contains a comma-separated list of five values.

ITEM	VALUE
Log pages	`True` or `false`
Time stamp log entries	A valid time format string to specify a time stamp, or `null`.
Log responses to test items	`True` or `false`
Log when a test item is locked	`True` or `false`
Compile score summary at the end of the log	`True` or `false`

EXAMPLES

```
ASYM_LogOptions of this book = "true,null,true,false,true"
```

SYNTAX

```
ASYM_LogSetOptions(<heading>,<log pages>,<log times>,<log responses>,
  <log on lock>,<log stats>)
```

DESCRIPTION

Specifies the options for the information that is recorded to the native (not DHTML) log.

RETURNS

If successful, the function returns `true`.

If any one of the parameters is invalid, the function returns `false`.

NOTES

Use this function *only* if you are calling `ASYM_LogStart()` to start the log rather than letting the system book read the `ASYM_ log` properties of the book.

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<heading>`	A string that specifies the information to be printed at the top of the log. This information is printed as soon as the `ASYM_LogStart()` function is called. The string can contain any valid expression, such as `%ASYM_GetSystemVar("s_StudentName")%`.
`<log pages>`	`True`, `false`, or `null`. If `true`, pages are logged whenever a page is entered (the `enterPage` message is sent). If `false` or `null`, pages are not logged.
`<log times>`	A valid `sysTimeFormat` string. Each log entry is time stamped using this `time` format. If `null`, no time information is recorded.
`<log responses>`	`True` or `false`. If `true`, responses are logged as they are recorded. This requires question objects to call `LogWriteEntry()` every time they detect a valid response. The system calls `LogWriteEntry()` automatically for question objects. If `null` or `false`, responses are not logged as they are recorded.
`<log on lock>`	`True` or `false`. If `true`, the state of question objects is recorded when the question is locked for any reason.
`<log stats>`	`True` or `false`. If `true`, summary statistics will be compiled and recorded at the end of the log when the log is closed with an `ASYM_LogStop()` call, or when leaving the application, whichever happens first.

EXAMPLES

```
to handle buttonClick
    SYSTEM logFName
    get ASYM_LogSetOptions("%ASYM_GetSystemVar("s_StudentName")%", \
      true,null,true,false,true)
    get ASYM_LogStart(logFName,false,false)
end
```

SYNTAX

```
ASYM_LogStart(<book>,<file name>,<append>,<encrypt>,<destination>,
  <log type>)
```

DESCRIPTION

Opens a log file and puts initial entries into it.

RETURNS

If successful, the function returns the log file name; otherwise, it returns `false`.

NOTES

The log file remains ready for recording until the `ASYM_LogStop()` function is called (see `ASYM_LogSetOptions()` for what is recorded).

The log file is actually closed between item recording to minimize the chance that data will be lost if a system problem occurs.

This function is called automatically by the runtime system book when it receives an `enterApplication` message and the `ASYM_LogType` property of the book is not `null`.

Only one log file can be active at any time.

Calling this function automatically closes any active log file.

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<book>`	The name of the book to be logged (usually `this book`).
`<file name>`	The name of the log file. If the name does not include a path, the system uses the default logging directory to store the log. If the file name is complete, the default log directory is ignored. If `null`, the system automatically creates a unique log file name.
`<append>`	`True` or `false`. If `true`, the log is `appended` to the specified `<file name>`. This parameter is ignored if `<file name>` is `null`. If `<append>` is `false` or `null`, but `<file name>` is not `null`, the system overwrites any existing log file with the same `<file name>`.
`<encrypt>`	`True` or `false`. If `true`, the text of the log file is encrypted to prevent tampering. The `encryption key` is the value of the system variable `s_ASYM_LogEncrytKey`. If this system variable is `null`, the encryption key is the value of the book property `ASYM_LogEncryptKey`. If `<encrypt>` is true, but the system does not find an encryption key value, it creates an encryption key based on the password "`sesame`". You can assign an encryption password to the book using the Book Properties - Instructor Options dialog box.
`<destination>`	The value of the `ASYM_LogDestination` property of the book. Its value is a string that contains one of the following values: `<sameDir>`, `<iniDir>`, `<tempDir>`, `<floppy>`, `<ask>`, `<fax>`, `<email>`, `<printer>`, `<special>`, or `null`.
`<log type>`	A string that contains one of the following values: "`transcript`", or "`responseData`". If this parameter is `null`, "`transcript`" is used.

EXAMPLES

```
to handle buttonClick
    SYSTEM logFName
    get ASYM_LogSetOptions("%ASYM_GetSystemVar("s_StudentName")%", \
      true,null,true,false,true)
    get ASYM_LogStart(logFName,false,false)
end
```

ASYM_LogStatus()

TB89R.SBK Logging Function

SYNTAX `ASYM_LogStatus()`

DESCRIPTION Specifies whether a log is active.

RETURNS If a log is currently active, the function returns the `name` and `size` of the log file. If no log is currently active, the function returns `null`.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS No parameters.

EXAMPLES

```
-- If logging is not on, turn it on
if ASYM_LogStatus() is null
    get ASYM_LogStart(LogFileName,false,false)
end
```

ASYM_LogStop()

TB89R.SBK Logging Function

SYNTAX `ASYM_LogStop(<file name>)`

DESCRIPTION Writes a final entry in the current log file, closes the file, and stops recording.

RETURNS If successful, the function returns the log `file name`; otherwise it returns `null`.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	The name of the log file. The name should match the name returned by `ASYM_LogStart()` or be `null`.

EXAMPLES

```
-- stop the logging if not already stopped
if ASYM_LogStatus() <> null
    get ASYM_LogStop()
end
```

ASYM_LogType

User Property

DESCRIPTION A book property that specifies the log type for a session.

NOTES The system checks this property when it handles the `enterApplication` message to determine if it should create a log for the current session.

VALUES A string that contains one of the following values: `none` (or `null`), `transcript`, or `responseData`.

EXAMPLES

```
ASYM_LogType of this book = "transcript"
```

ASYM_LogWriteEntry()

TB89R.SBK Logging Function

SYNTAX `ASYM_LogWriteEntry(<text>)`

DESCRIPTION Writes an entry in the current log file. Depending on current log options, the entry may be encrypted.

RETURNS If successful, the function returns `true`; otherwise, it returns `false`.

This function always returns `false` if there is no current log. Call `ASYM_LogStatus()` to verify if a current log exists.

`ASYM_LogWriteEntry()` does not add a `CRLF` at the end of the entry. If you want your entries to appear on separate lines you should append a `CRLF` to each entry.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<text>`	The text of the entry to write to the log.

EXAMPLES

```
--Incorporates a student comment into the log
txt = ASYM_trim(text of field "Comment")
if txt <> null and ASYM_LogStatus() <> null
    get ASYM_LogWriteEntry(txt & CRLF)
end
```

ASYM_NextNavigablePage()

TB89R.SBK Navigation

SYNTAX `ASYM_NextNavigablePage()`

DESCRIPTION Gets the page reference of the nearest page (after the current page) with the `skipNavigation` property set to `false`.

RETURNS A page reference of the next navigable page.

If no navigable page is found or if the next navigable page is the current page, the function returns `null`.

NOTES This function steps through the book's pages until it finds a valid page. To avoid performance degradation, use this function sparingly in large books that contain many pages with `skipNavigation` set to `true`.

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

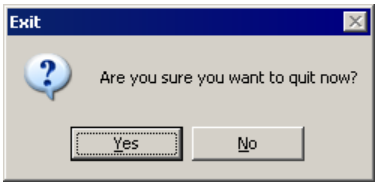
PARAMETERS No parameters.

EXAMPLES

```
-- Handler in script of "Next page" button
notifyBefore enterPage
    my enabled = (ASYM_NextNavigablePage() <> null)
end
```

SYNTAX `ASYM_MessageBox(<message>,<caption>,<icon>,<buttons>,<TopicID>,<File>)`

DESCRIPTION Displays a standard Windows dialog message box containing the message, caption, icon and buttons specified. This function is essentially a wrapper around the Windows `MessageBox()` API function.



RETURNS A value corresponding to the dialog button the user clicked.

Possible return values are: `OK`, `cancel`, `abort`, `retry`, `ignore`, `yes`, `no`.

NOTES This dialog box is somewhat limited in functionality. It is recommended that you use `ASYM_Request()` instead, which has many more features to offer.

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<message>`	The text message that will appear in the dialog.
`<caption>`	The text to be displayed in the caption of the dialog.
`<icon>`	The type of icon to be displayed to the left of the message. The valid values are: `stop`, `hand`, `question`, `exclamation`, `warning`, `information`, `asterisk`.
`<buttons>`	Determines the buttons that will be included in the dialog. Valid values are: `OK`, `OKCancel`, `RetryCancel`, `AbortRetryIgnore`, `YesNo`, `YesNoCancel` It is not permitted to specify your own custom caption text.
`<TopicID>`	A Help context number, a Help key string, or `null`. This value is passed to the `ASYM_WinHelp()` function if the user presses the `F1` key in the message box.
`<File>`	The name of the `WinHelp` file to use. If `null`, a file name is constructed by getting the current book's file name and changing the extension to `.HLP`. This value is passed to the `ASYM_WinHelp()` function if the user presses the `F1` key in the message box.

EXAMPLES

```
msg  = "Are you sure you want to quit now?"
cap  = "Exit"
icn  = "question"
btns = "YesNo"
answ = ASYM_MessageBox(msg,cap,icn,btns)
```

ASYM_Modal

User Property

DESCRIPTION A book property that specifies if ToolBook runs in an `exclusive mode` that does not allow users to switch to another application using the `Alt+Tab` Windows shortcut.

VALUES `True` or `false`. If `true`, ToolBook runs in `exclusive mode`.

ASYM_ObjectsWhere()

TB89R.SBK Object Function

SYNTAX `ASYM_ObjectsWhere(<owner>,<object types>,<expression>)`

DESCRIPTION Returns a list of objects that meet the criteria defined in the specified expression.

NOTE This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<owner>`	A valid reference to the parent object of the objects to be examined.
`<object types>`	A list of one or more ToolBook object types, or `null`. If `null`, all object types are examined.
`<expression>`	A logical expression that returns `true` for the specified object types. In this expression, the variable `IT` represents the object being examined. For example, "`name of IT <> null`".

EXAMPLES

```
objList = ASYM_ObjectsWhere(this page,"field", "activated of IT is false")
```

ASYM_Offset()

TB89R.SBK String Functions

SYNTAX `ASYM_Offset(<string>,<source>[,<startpos>])`

DESCRIPTION Locates the first occurrence of `<string>` in `<source>`. This function is identical to the `offset()` function except this ASYM version allows for an optional starting position.

RETURNS Returns the character position in which `<string>` was found in `<source>`. If `<string>` cannot be found within `<source>` then `0` is returned.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<string>`	An expression that results in a string value.
`<source>`	An expression that results in a string value.
`<startpos>`	Optional Parameter. An expression that results in a number indicating the character position to start the search at.

EXAMPLES

```
to handle buttonClick
    x = text of field "partNumber"
    pos = ASYM_Offset("#",x,2)
    char pos of x = "@"
    text of field "partNumber" = x
end
```

ASYM_ParsePath()

TB89R.SBK File Function (Filename)

SYNTAX `ASYM_ParsePath(<path>,<operation>)`

DESCRIPTION Use this function to get the component parts from a full or partial path string.

RETURNS The desired segment of a full or partial path.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION	
`<path>`	A path to parse.	
`<operation>`	`Disk`	Returns the disk drive letter followed by a colon.
	`Extension_only`	Returns the file extension without the period.
	`Filename`	Returns the filename with its file extension.
	`Filename_only`	Returns only the filename without its file extension.
	`Path`	Returns the full disk drive and directory with no filename (including the trailing backslash character).

EXAMPLES

```
-- when path = "c:\Program Files\My Application\My File.txt"
put ASYM_ParsePath(path,"Disk")           -- displays C:
put ASYM_ParsePath(path,"Extension_only") -- displays txt
put ASYM_ParsePath(path,"Filename")       -- displays My File.txt
put ASYM_ParsePath(path,"Filename_only")  -- displays My File
put ASYM_ParsePath(path,"Path")           -- displays C:\Program
                                             Files\My Application\
```

ASYM_PopGlossary()

TB89R.SBK Glossary Function

SYNTAX `ASYM_PopGlossary(<term>,<book>,<style>,<position>,<caption>,<cursor>)`

DESCRIPTION Finds a glossary page and displays it in a popup window.

By default, the popup window is `shadowed` and can be closed by any click onscreen.

RETURNS `True` if the glossary term could be found, otherwise returns `false`.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<term>`	The term to find. This should be the name of a glossary page.
`<book>`	Optional argument specifying the book in which to find the term. If this is a pathless file name, the system attempts to locate the book in the default `Hyperlinks` path. If this parameter is `null`, the `ASYM_GlossaryName` property of the current book is checked. If that property is `null`, the system tries to locate the term in a background named "`Glossary`" of the current book. If these tests fail, the default book name "`GLOSSARY.TBK`" is used.

PARAMETER	DESCRIPTION
<style>	An optional style setting for the popup window, or null. If null, the default is "shadowAutoClose". "shadow" "shadowAutoClose" "thickFrame" "thinFrame" "dialogFrame" If you set <style> to shadow, you must include a script to close the window. For other styles, the interface provides a method for the user to close the window without further scripting.
<position>	An optional position in page units. If null, the page pops up as close as possible to the mouse position while still displayed fully on the screen.
<caption>	An optional caption for the viewer in which the page is displayed. The caption is ignored if the viewer style does not include a caption bar.
<cursor>	An optional sysCursor number, or null. The cursor is displayed while the popup window is being created.

EXAMPLES

```
get ASYM_PopGlossary("rope","terms.tbk","thickFrame",null,"Terms",null)
```

ASYM_PopGlossaryStyle

User Property

DESCRIPTION

Works in conjunction with ASYM_AutoGlossary or ASYM_AutoHotwords to specify the style of the popup window.

VALUES

shadow, shadowAutoClose, thickFrame, thinFrame, dialogFrame.

The default is shadow.

ASYM_PopRTFHelp()

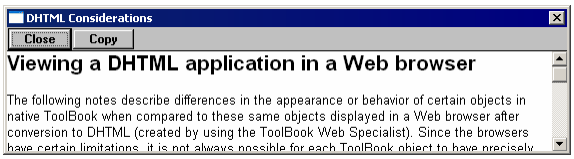
TB89R.SBK Dialog Box

SYNTAX

```
ASYM_PopRTFHelp(<file name>,<caption>)
```

DESCRIPTION

Displays a .TXT or .RTF file in a popup window, complete with a Copy and Close button. The dialog can even be resized by the user.



RETURNS

If the file is found, the function returns true, otherwise it returns false.

NOTES

If displaying RTF note that ToolBook's RTF support is not as robust as many other products such as Microsoft Word. This may result in ToolBook not displaying all the RTF features that the file contains.

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<file name>	The name and path of a .TXT or .RTF file with text to be displayed in a popup window.
<caption>	A string of text that specifies the caption of the popup window.

EXAMPLES

```
get ASYM_PopRTFHelp("c:\project\readme.rtf","Release Notes")
```

ASYM_PreviousNavigablePage()

TB89R.SBK Navigation

SYNTAX

```
ASYM_PreviousNavigablePage()
```

DESCRIPTION

Gets the page reference of the nearest page (before the current page) with the skipNavigation property set to false.

RETURNS

A page reference of the previous navigable page.

If no navigable page is found or if the previous navigable page is the current page, the function returns null.

NOTES

This function steps through the book's pages until it finds a valid page. To avoid performance degradation, use this function sparingly in large books that contain many pages with skipNavigation set to true.

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

No parameters.

EXAMPLES

```
-- Handler in script of "Previous page" button
notifyBefore enterPage
    my enabled = (ASYM_PreviousNavigablePage() <> null)
end
```

ASYM_RandomList()

TB89R.SBK String Functions

SYNTAX

```
ASYM_RandomList(<number>)
ASYM_RandomList(<number>,<items>)
```

DESCRIPTION

This function generates a random list of numbers.

This function also can generate a random list of list items.

RETURNS

If the <items> parameter is not passed or is null, the function will take the number passed in the <numbers> parameter and return a randomly generated comma separated list containing values from 1 to <number>. This return value will not contains duplicates of other values in the list.

If the <items> parameter is used and not null, this function will take the number passed in the <numbers> parameter and return a randomly generated comma separated list containing items from <items>. This return value will not contains duplicates of other values in the list, unless your original list contained repeats.

NOTES

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<number>	An expression that results in a number indicating how many items to select.
<items>	Optional Parameter. A comma-separated list of items.

EXAMPLES

```
-- generate a simulated shuffled deck of cards
cardOrder = ASYM_RandomList(52)

-- which days do I work this week
x = "Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday"
schedule = ASYM_RandomList(5,x)
```

ASYM_ReplaceFileExtension()

TB89R.SBK File Function (Filename)

SYNTAX

```
ASYM_ReplaceFileExtension(<file name>,<file extension>)
```

DESCRIPTION

This function can be used to add a file extension to a file name, replacing the current extension if it has one.

RETURNS

The file name with the new extension.

NOTES

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<file name>	The file name you want to add a file extension to.
<file extension>	The file extension you want to add to the filename.

EXAMPLES

```
-- Create a log filename based on the book's name
x = name of this book
fileName = ASYM_ReplaceFileExtension(x,"log")
```

ASYM_Request()

TB89R.SBK Dialog Box

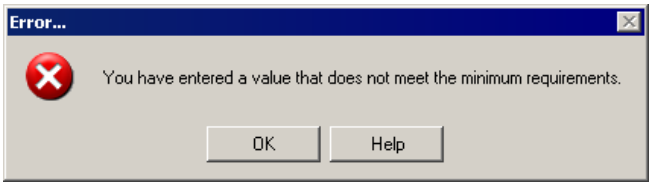
SYNTAX

```
ASYM_Request(<caption>,<text>,<icon>,<buttons>,<width>,<position>,
             <frame>,<checkbox>,<radio>,<help>)
```

DESCRIPTION

Displays a modal dialog which prompts the user to respond to a message by clicking on an available button.

This function provides a more flexible alternative to using the `Request` command.



RETURNS A value of `NULL` is returned if the user does one of the following: press `ESC`, close the dialog using the `X` on the caption bar, or press the designated `Cancel` button.

Note: The `ESC` key will not function, and the `X` on the caption bar will not function, if you have not designated a `Cancel` button as described in the `<buttons>` parameter description.

Otherwise, the following 3 textlines are returned, separated by `CRLFs`:

```
<button>
<radio button>
<checkbox buttons>
```

RETURN VALUE	DESCRIPTION
`<button>`	A number representing which button was pressed. This number corresponds to the matching textline number passed in the `<buttons>` parameter.
`<radio button>`	A number representing which radio button was checked. This number corresponds to the matching textline number passed in the `<radio>` parameter. This line will be included even if this feature was not utilized or even if no radio button was checked, which would result in an empty line.
`<checkbox buttons>`	A number representing which checkbox buttons were checked. This commas separated list of numbers corresponds to the matching textline numbers passed in the `<checkbox>` parameter. This line will be included even if this feature was not utilized or even if no checkbox buttons were checked, which would result in an empty line.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<caption>`	The text to be applied as the caption of the dialog box. Maximum character length is `78`. If `null`, no caption will appear.
`<text>`	Text to display in the body of the dialog box. `RichText` is permitted. If `null`, no text will appear.
`<icon>`	Valid values are: `stop`, `question`, `exclamation`, `information` You may instead pass a reference to an `icon resource`, such as `icon "Apple" of this book` If `null` no icon will be displayed in the dialog box.

PARAMETER	DESCRIPTION
<buttons>	There are two different methods for defining which buttons captions to display: STANDARD: Valid values are: OK, OKCancel, RetryCancel, AbortRetryIgnore, YesNo, YesNoCancel CUSTOM: A list of textlines separated by CRLF characters that contain up to ten button captions. These buttons will appear in a single row as command buttons just above the base of the dialog. If null, a single OK button will be used. The following special characters (use one or more) can be defined at the beginning of each caption, and will not appear in the actual caption text: + Indicates that this button will be the Default button. ? Indicates that this button will be the Help button. If used, pressing the button or pressing F1 will open the specified help file. x Indicates that this button will be used as the Cancel button. If used, ESC and the X in your caption bar will trigger this cancel button. (Indicates that this button will be disabled when the dialog is shown. / Indicates that all subsequent characters are part of the button's caption. This allows you to use one of the above characters as the first real character of the caption. [example: if your caption needs to read +5%, to protect the + character from being interpreted as a special character, write it as /+5%]
<defText>	Default text to be displayed in a single line editable text field. This will be positioned directly below <text>. RichText is not permitted.
<width>	The width (in pixels) of the dialog. If null the width of the dialog will be determined automatically. This value may be overridden if found to be too small (less than 200 pixels) or if less than the width of <buttons>, <radio> or <checkbox>.
<position>	The position (in pixels) of the upper left corner of the dialog. If null the position of the dialog will be centered on your screen.
<frame>	If you would like an 3D style inset frame to be shown surrounding the elements of the dialog, specify true, otherwise specify false. If null the frame will not be shown.
<radio>	A CRLF separated list of textlines containing up to 10 radio button captions. These buttons will appear above <buttons> stacked vertically and left justified with <text>. The radio buttons will behave in a mutually exclusive manner. If null, no radio buttons will appear. The following special characters (use one or more) can be defined at the beginning of each caption, and will not appear in the actual caption text: + Indicates that this button will be the Default button. ? Indicates that this button will be the Help button. If used, pressing the button or pressing F1 will open the specified help file. x Indicates that this button will be used as the Cancel button. If used, ESC and the X in your caption bar will trigger this cancel button. (Indicates that this button will be disabled when the dialog is shown. / Indicates that all subsequent characters are part of the button's caption. This allows you to use one of the above characters as the first real character of the caption. [example: if your caption needs to read +5%, to protect the + character from being interpreted as a special character, write it as /+5%]

PARAMETER	DESCRIPTION
<checkbox>	A CRLF separated list of textlines containing up to 10 checkbox button captions. These buttons will appear above <buttons>, and below <radio>, stacked vertically and left justified with <text>. If null, no checkbox buttons will appear. The following special characters (use one or more) can be defined at the beginning of each caption, and will not appear in the actual caption text: + Indicates that this button will be the Default button. ? Indicates that this button will be the Help button. If used, pressing the button or pressing F1 will open the specified help file. x Indicates that this button will be used as the Cancel button. If used, ESC and the X in your caption bar will trigger this cancel button. (Indicates that this button will be disabled when the dialog is shown. / Indicates that all subsequent characters are part of the button's caption. This allows you to use one of the above characters as the first real character of the caption. [example: if your caption needs to read +5%, to protect the + character from being interpreted as a special character, write it as /+5%]
<help>	Allows one of the <buttons> to open a help file by either clicking the button or pressing F1. This parameter is a comma separated list that can contain up to two items. Item 1 The name of the WinHelp file to use. If null, a file name is constructed by getting the current book's file name and changing the extension to .HLP. Otherwise, the file name must contain a path, or the file must be in the DOS path. Item 2 Optional. The Help Topic or Help Topic ID to look up. If null, the Contents tab will be opened. If null, no button will function as a Help button.

EXAMPLES

```
example_caption  = "Error..."
example_text     = "You have entered a value that does not " & \
                   "meet the minimum requirements."
example_icon     = icon "bad data" of this book
example_buttons  = "OK" & CRLF & "?Help"
example_width    = null
example_position = null
example_frame    = false
example_check    = null
example_radio    = null
example_help     = "c:\tutor\tutor.hlp,447"
reply = ASYM_Ask(example_caption,example_text,example_icon,      \
                 example_buttons,example_width,example_position, \
                 example_frame,example_check,example_radio,      \
                 example_help)
```

ASYM_ResetPosition

User Property

DESCRIPTION A property of graphic and draw objects that specifies the position that an object returns to when the ASYM_Reset message is sent to the page.

NOTES The object's script must contain the handler that moves the object.

VALUES A list of two numbers that specifies the location of the upper-left corner of the object on the page.

ASYM_SetCurrentDirectory()

TB89R.SBK File Function (Directory)

SYNTAX `ASYM_SetCurrentDirectory(<path>)`

DESCRIPTION Sets the current drive and directory to the specified path.

RETURNS If successful, the function returns `1`.

Otherwise, it returns `-1` or `-3` to indicate the specified drive is invalid.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<path>`	Specifies the path for the new directory. The directory specification can end in a backslash (\).

EXAMPLES

```
oldDir = ASYM_CurrentDirectory()
send saveAs
get ASYM_SetCurrentDirectory(oldDir)
```

ASYM_ShortFileName()

TB89R.SBK File Function (Filename)

SYNTAX `ASYM_ShortFileName(<file specification>)`

DESCRIPTION Removes the path portion of `<file specification>` and returns the file name.

Use this function to remove drive and directory dependencies from OpenScript references.

RETURNS A file name (without drive and directory information).

For example, if your file specification is: `c:\projects\fire safety\course1.txt`
The function would return just: `course1.txt`

NOTES `ASYM_ParsePath()` can also provide this functionality.

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<file specification>`	A complete file name, or a URL that ends in a file name.

EXAMPLES

```
request ASYM_ShortFileName(name of this book)
```

ASYM_StringOf()

TB89R.SBK String Functions

SYNTAX `ASYM_StringOf(<string>,<numberOfCopies>)`

DESCRIPTION This function takes your original string value and gives it back to you repeated several times.

RETURNS Returns your original string repeated the number of times specified by `<numberOfCopies>`.

Returns `null` if `<numberOfCopies>` is not a positive number.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<string>	An expression that results in a string value.
<numberOfCopies>	An expression that results in a number.

EXAMPLES

```
dividerBar = ASYM_StringOf("-",60)
```

ASYM_TempDir()

TB89R.SBK File Function (Temp)

SYNTAX

```
ASYM_TempDir()
```

DESCRIPTION Gets the directory in which Windows applications store temporary files for the currently logged in user.

RETURNS A complete [short filename] directory specification ending with a backslash (\).

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS No Parameters.

EXAMPLES

```
tempFolder = ASYM_TempDir()
```

ASYM_TextlineFromPos()

TB89R.SBK String Functions

SYNTAX

```
ASYM_TextlineFromPos(<position>,<text>)
```

DESCRIPTION Locates the textline containing the specified character position.

RETURNS Returns a number representing the textline number in which the specified character position exists.

If you specify a `<position>` greater than the total characters in `<text>` this function will return the `textlineCount()` of `<text>`.

If any error occurs then the function return `0`. This could occur if `<position>` does not contain a valid whole number.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<position>	An expression that results in a whole number representing the character position.
<text>	An expression that results in a string value.

EXAMPLES

```
-- Display the textline which contains the spelling error
to handle buttonClick
    x = text of field "sample"
    tlNum = ASYM_TextLineFromPos(offset(badWord,x),"Wensday")
    request textline tlNum of x
end
```

ASYM_TextlineInList()

SYNTAX `ASYM_TextlineInList(<textline>,<list>)`

DESCRIPTION Determines if the specified `<textline>` exists in a `CRLF` separate `<list>` of text lines. This function is equivalent to saying: `(textlineOffset(x,y) <> 0)`

RETURNS The function returns `true` if the textline is in the list, and `false` if the textline is not in the list.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<textline>`	An expression that results in a string value.
`<list>`	An expression that results in a `CRLF` list of textlines.

EXAMPLES

```
to handle buttonClick
    possibleAnswers = "apple" & CRLF & "banana" & CRLF & "pear"
    ask "What is one of my favorite fruits?"
    answ = IT
    if ASYM_TextlineInList(answ,possibleAnswers)
        request "You are correct."
    else
        request "Sorry, incorrect!"
    end
end
```

ASYM_TextLineOffset()

SYNTAX `ASYM_TextLineOffset(<line>,<text>[,<startpos>])`

DESCRIPTION Locates the first occurrence of a textline in a string. This function is identical to the `textlineOffset()` function except this ASYM version allows for an optional starting position.

RETURNS Returns the textline position in which `<line>` was found in `<text>`. If `<line>` cannot be found within `<text>` then `0` is returned.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<line>`	An expression that results in a string value.
`<text>`	An expression that results in a string value.
`<startpos>`	Optional Parameter. An expression that results in a number indicating the textline position to start the search at.

EXAMPLES

```
to handle buttonClick
    lst = "apple" & CRLF & "banana" & CRLF & "orange" & CRLF & "apple"
    ask "What are you looking for?"
    answ = IT
    pos = itemOffset(answ,lst,2) -- looking for 2nd occurrence
    if pos > 0
        request "Yes, that item is in textline position" && pos
    else
        request "Sorry, didn't find a match"
    end
end
```

ASYM_TextToPrinter()

TB89R.SBK Printing Function

SYNTAX `ASYM_TextToPrinter(<string>,<options>,<window handle>,<bShowDlg>)`

DESCRIPTION Prints a text string to the current default printer.

This function automatically wraps the text to fit on the printed page, with fixed (8-space) tabs. Only one style of formatting is available for each document.

RETURNS If successful, the function returns `true`.

Otherwise, it returns `false`.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<string>`	The text string to print.
`<options>`	A list of textlines (`CRLF`-separated) that specify the following options: `Document Name` — A name for the document to be printed. `Font Face` — A valid font name. `Font Style list` — Bold, italic, underline, or strikeout. `Font Size` — A valid font size. `Margins in inches` — A string that contains a list of four numbers that correspond to the `<left>`, `<top>`, `<right>`, and `<bottom>` margins.
`<window handle>`	The window handle of the parent window of the Cancel dialog box.
`<bShowDlg>`	`1` Shows the Cancel dialog box, `0` Does not show the Cancel dialog box.

EXAMPLES

```
to handle buttonClick
    txt = text of field "sample"
    docName = "Test"
    fontF = "Times New Roman"
    fontS = "Bold"
    fontSz = 10
    marg = "1.5,1,1.5.1"
    opts = docName & CRLF & fontF & CRLF & fontS & \
           CRLF & fontSz & CRLF & marg
    hndl = windowHandle of mainWindow
    showDlg = 1
    get ASYM_TextToPrinter(txt,opts,hndl,showDlg)
end
```

ASYM_Ticks()

TB89R.SBK Timer Function

SYNTAX `ASYM_Ticks()`

DESCRIPTION This function retrieves the current system time in milliseconds, which is the time elapsed since Windows started.

It is useful for determining how long something takes, for example, the time it takes a user to respond to a question.

RETURNS The current system time in `milliseconds`.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS No Parameters.

EXAMPLES

```
to handle buttonClick
    system oldTicks
    if oldTicks = null
        oldTicks = ASYM_Ticks()
    end
    curTicks = ASYM_Ticks()
    if curTicks - oldTicks < 2000
        request "You are not allowed to click more than once every 2 seconds."
    else
        send ProcessAgain
        oldTicks = curTicks
    end
end
```

ASYM_TpID

User Property

DESCRIPTION A background property that specifies a unique `ID` for a background.

NOTES The value is stored separately from the background's `ID` number and is set automatically by the authoring system book (`TB89A.SBK`) when you create a new background.

This property identifies backgrounds that are identical when you merge books or import templates without having to rely on a central registry for backgrounds. Because this property is likely to have different values for any two backgrounds, the New Background dialog box uses it to indicate which layout templates match backgrounds already in the book, and to warn you if you are about to import a layout template that is already in use in your book. You can also use this property to check whether pages from different books actually belong to the same background prior to merging the books.

VALUES A large random string of numbers, or `null`.

ASYM_Trim()

TB89R.SBK String Functions

SYNTAX `ASYM_Trim(<string>`

DESCRIPTION This function takes the string and removes all white space that occurs at the beginning and end of your string.

RETURNS Returns the modified string.

White space is defined as any nonprintable character which includes space, tab, `CRLF` and all other characters with `ANSI` values less than `33`. Typically used to remove unnecessary spaces from the beginning and end of a string value.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<string>`	An expression that results in a string value.

EXAMPLES

```
to handle buttonClick
    ask "What is your name?"
    uName = ASYM_Trim(IT)
    request "Hello," && uName & ". Welcome to the seminar."
end
```

ASYM_ViewerContainer()

TB89R.SBK Viewer Function

SYNTAX `ASYM_ViewerContainer(<object reference>)`

DESCRIPTION Returns the ToolBook viewer in which the object is being displayed.

RETURNS A viewer reference to the viewer in which the object is displayed.

In the event that the object is displayed in more than one viewer, the function will prioritize the return value, based on the following viewer references:

```
1   targetWindow
2   focusWindow
3   ASYM_WorkWindow
4   mainWindow
```

If none of these viewers are represented in the list of viewers containing the object, the first viewer on the list will be returned.

Returns the viewer parent of the target object if the `<object reference>` parameter is `null`.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<object reference>`	A valid ToolBook object reference.

EXAMPLES

```
myViewer = ASYM_ViewerContainer(button "Next" of background "Content")
```

ASYM_Wait()

TB89R.SBK Timer Function

SYNTAX `ASYM_Wait(<milliseconds>,<keysThatBreak>)`

DESCRIPTION Yields to other processes and monitors keyboard and mouse events for the cue to interrupt the wait.

A keystroke or a mouse click anywhere in the system will cause the wait to be interrupted, as long as the corresponding virtual key is listed for its `<keysThatBreak>` parameter or one of the default parameters is used.

RETURNS The key constant for the key that was pressed, if any (includes mouse buttons).

If `<keysThatBreak>` was `0` and a key or mouse button was pressed during the wait, the function returns `255`.

If no key was pressed by the time the wait time elapsed, the function returns `0`.

If the function could not be executed, the function returns `-1` (typically because of invalid parameters).

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
<milliseconds>	The number of milliseconds to wait.
<keysThatBreak>	A list of ANSI values for typed characters on which the wait must be interrupted. If null, the wait can be interrupted by pressing Escape. If 0, the wait can be interrupted by any keyboard or mouse click.

EXAMPLES

```
get ASYM_Wait(5000,0)
if IT = 255
    send next
else
    send playAttractorAnimation
end
```

ASYM_WID_AnsArray

Extended Property

DESCRIPTION A question object property that defines the settings for responses and feedback options.

VALUES A two-dimensional array containing all the information the system requires to respond to a collection of answers that are defined for the question object.

Each row corresponds to a different answer.

The following table describes each element of the array.

ELEMENT	DESCRIPTION
[1] Answer Identifier	The name of an answer as it appears in the object's Question Properties dialog box.
[2] Hyperlink Page Name	The name of the destination page. If no name, "ID <number>" is used. This is the primary identifier used by the system when attempting to locate the page. If it fails, the secondary identifier is used. The following special page constants are used to reference relative pages: <next>, <previous>, <first>, <back>, or <last>.
[3] Hyperlink Alternate Page Identifier	"ID <number>", or the caption of the page (if defined). This field can be used when the caption of a page becomes available (if the field contains a caption and not an ID) to display the caption rather than the names of pages in the hyperlinking user interface.
[4] Hyperlink Book Name	By default, this is a file name (path not included). The ASYM_HyperPath property of the book is used to specify the search path. Alternatively, a fully-qualified path can be included. If the value of this element is null or <this book>, the current book is used.
[5] Hyperlink	(Reserved for future use.)
[6] Hyperlink Transition	A transition effect string ready for use by OpenScript. If element [7] is set to jump, element [6] can be an OpenScript effect string. If element [7] is set to popup, element [6] can be set to one of the following special popup viewer type constants: dialogFrame, popup, popupAutoclose, standard or standardAutoclose.
[7] Hyperlink Type	Possible values are jump, popup, or null. Currently defined as jump. If null, signifies that no hyperlink transition will occur.
[8]	(Reserved for future use.)
[9]	Specifies the system uses an hourglass cursor when navigating.

ELEMENT	DESCRIPTION
[10] Correct	If `true`, this answer is tagged as correct.
[11] Scoring Weight	The scoring weight of the answer. Normally a number between `0` and `1`. When computing the score, the weights for all answers are added and normalized so that their total is the same as the maximum score value assigned to the question.
[12] Feedback Text	A literal string.
[13] Feedback Clip	The name of a media clip.
[14] Feedback Text Option	If `true`, the feedback text is shown only when the feedback clip fails or is not available. `Null` is the same as `false`.
[15] Feedback OpenScript Message	The OpenScript message with optional parameters to be sent to the question object.
[16]	(Reserved for future use.)
[17]	Specific options that depend on the type of question object.

ASYM_WID_AnswerLocked

Extended Property

DESCRIPTION An extended property that specifies if the question is locked.

NOTES Use this property to prevent a student from answering a question under certain conditions.

VALUES `True` or `false`.

If `true`, the question is locked.

ASYM_WID_Author

Extended Property

DESCRIPTION An extended property that specifies the name of the organization that created the selected object in the Catalog.

NOTES For all the standard widgets in the catalog, this value will be set to "`click2learn.com`".

VALUES A string of text.

ASYM_WID_AutoLockAnswer

Extended Property

DESCRIPTION An extended property that specifies if the question is automatically locked after it is answered.

VALUES `True` or `false`.

If `true`, the question is locked after being answered.

ASYM_WID_AutoReset

Extended Property

DESCRIPTION An extended property that specifies if a question object is automatically reset when the page receives the `enterPage` or `leavePage` message.

NOTES To reset all the question objects in a book, send the `ASYM_Reset` message to the book.

Set `ASYM_WID_Reset` to `false` or `null` if you want to tally the score at the end of a session. The system clears question object scores and response settings when the objects reset.

VALUES `Null` or `false`, `enterPage`, `leavePage`, or `true`.

VALUE	DESCRIPTION
`Null` or `false`	Question is never reset.
`enterPage`	Question is reset when the page receives the enterPage message.
`leavePage`	Question is reset when the page receives the leavePage message.
`true`	Question is automatically reset on both enterPage and leavePage.

EXAMPLES `ASYM_WID_AutoRest of group "Multiple Choice" = "leavePage"`

ASYM_WID_Correctness

Extended Property

DESCRIPTION The current degree of "correctness" of an Instructor question object, calculated by adding the score weights for the current response or responses.

NOTES You cannot set this property.

VALUES The value of `ASYM_WID_Correctness` is a number between `0` ("Incorrect") and `1` ("Correct").

You can get this property regardless of whether the option to score the question is set or not.

ASYM_WID_CreateDate

Extended Property

DESCRIPTION An extended property that specifies the creation date for the selected object in the Catalog.

NOTES For all the standard objects in the catalog, this value will be set to a date representing the day the object was first implemented.

VALUES A string of text.

ASYM_WID_DelayFeedback

Extended Property

DESCRIPTION An extended property that specifies if feedback plays immediately after a user answers a question.

VALUES `True` or `false`.

If `false`, question feedback plays immediately after the user answers the question.

ASYM_WID_Description

Extended Property

DESCRIPTION An extended property that specifies the description of the selected object in the Catalog.

NOTES When you add an object to the Catalog, or modify an existing object, you can add or revise this text to describe the object.

This property is **not** supported in ToolBook version 6.5 or higher. It has been replaced with the `Info_Description` property.

VALUES A string of text.

ASYM_WID_Doc

Extended Property

DESCRIPTION An extended property that specifies the file that runs when help is requested for the object.

NOTES The file that you associate to be the help file does not have to be a .HLP file. It can be any file you like, such as a .TXT file.

VALUES A valid file name.

ASYM_WID_DragSnap

Extended Property

DESCRIPTION An extended property that specifies if objects dropped on a question snap to the center of the question object's bounds.

VALUES `True` or `false`.

If `true`, the position of the object snaps to the center of the question object's bounds.

ASYM_WID_Editor

Extended Property

DESCRIPTION An extended property that specifies which book is used for editing properties of catalog objects when the Extended Properties or Question menu command is chosen from the Object Properties submenu on the Object menu.

NOTES The book is displayed in a modal viewer.

VALUES A valid file name of a book (path not included).

ASYM_WID_IsScored

Extended Property

DESCRIPTION An extended property that specifies if the question score is evaluated and included in the score that is tallied for the log or the CMS.

VALUES `True` or `false`.

If `true`, the score is evaluated and logged.

ASYM_WID_MaxScore

Extended Property

DESCRIPTION An extended property that specifies the maximum possible score for a question.

NOTES This value represents the total number of points the question is worth.

VALUES A string that contains a number.

ASYM_WID_MultipleAnswers

Extended Property

DESCRIPTION An extended property that specifies if a question allows multiple answers.

NOTES This is typically used for a multiple choice question object.

VALUES `True` or `false`.

If `true`, a question can have more than one correct response.

ASYM_WID_ReadyToRun

Extended Property

DESCRIPTION An extended property that specifies if a question object is ready to be used at `Reader` level.

NOTES When an object is ready, all of its properties have been set, such as those that control randomization, appearance, and movement.

ToolBook automatically clears this property when the question is reset to a non-ready state.

VALUES `True` or `false`.

If `true`, the question is ready to run at `Reader` level.

ASYM_WID_RejectWrong

Extended Property

DESCRIPTION An extended property that specifies if the question rejects incorrect responses.

NOTES If the incorrect response is dragged, it snaps back to its original position when rejected.

VALUES `True` or `false`.

If `true`, the question rejects incorrect responses.

ASYM_WID_TargetObject

Extended Property

DESCRIPTION An extended property that specifies the object to be used as a question target.

VALUES A string that contains a valid object reference.

If the object is a timer or a score button, the value can be an object reference, or `<all>`.

ASYM_WID_TimeChosen

Extended Property

DESCRIPTION An extended property that specifies the time when the user chooses the answer for a question with a time limit.

VALUES A string that contains a number representing the time in milliseconds.

ASYM_WID_TimeMax

Extended Property

DESCRIPTION An extended property that specifies the maximum time allowed for the user to answer a question with a time limit.

VALUES A string that contains a number representing the time in milliseconds.

ASYM_WID_TimeStart

Extended Property

DESCRIPTION An extended property that specifies the time when a question is started or reset.

VALUES A string that contains a number representing the time in milliseconds.

ASYM_WID_TimeUsed

Extended Property

DESCRIPTION An extended property that specifies the total amount of time used for a question with a time limit.

VALUES A string that contains a number representing the time in milliseconds.

ASYM_WID_TriesMax

Extended Property

DESCRIPTION An extended property that specifies the maximum number of tries allowed with a question object.

VALUES A string that contains a number.

ASYM_WID_TriesUsed

Extended Property

DESCRIPTION An extended property that specifies the number of tries used with a question object.

VALUES A string that contains a number.

ASYM_WinHelp()

TB89R.SBK Help Function

SYNTAX `ASYM_WinHelp(<topic>,<file name>)`

DESCRIPTION Calls the Windows function `WinHelp()` to display Help with the standard Windows Help engine. The function requires a Help file in `.HLP` format.

RETURNS `True` if Windows reports that the function is successful; otherwise `false`.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<topic>`	A `Help context number`, a `Help key string`, or `null`. If `<topic>` is a number, WinHelp is called with the `HELP_CONTEXT` flag. If `<topic>` is a key string, WinHelp is called with the `HELP_PARTIALKEY` flag. If `<topic>` is `null`, WinHelp is called with the `HELP_CONTENTS` flag.
`<file name>`	The file name and path of the WinHelp file to use. If `null`, a file name is constructed by getting the current book's file name and changing the extension to `.HLP`.

EXAMPLES

```
get ASYM_WinHelp("hose","FireSafety.HLP")
```

ASYM_WindowsDirectory()

TB89R.SBK File Function (Directory)

SYNTAX `ASYM_WindowsDirectory()`

DESCRIPTION Locates the current Windows directory in a format ready for concatenating with a file or directory name; for example, `c:\windows\`

RETURNS The current Windows directory.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS No Parameters.

EXAMPLES
```
winDir = ASYM_WindowsDirectory()
```

ASYM_WorkWindow()

TB89R.SBK Viewer Function

SYNTAX `ASYM_WorkWindow()`

DESCRIPTION Gets a valid `handle` to the viewer that is activated when called from a viewer that belongs to a current system book (such as a navigation tool bar or a dialog box implemented as a modal window).

The `handle` is returned for the viewer that is the last activated viewer that does not belong to a system book.

Most of the time, the value is the same as the `mainWindow` property, but it can be another viewer. The return value of this function is certain to be a valid viewer.

RETURNS A valid `viewer` reference.

NOTES This function checks and, if necessary, resets the system variable s_`ASYMWorkWindow`. If what appears to be the working window belongs to a system book, this function defaults to `mainWindow`. Use this function, rather than using s_`ASYMWorkWindow` directly, because the internal reference stored in s_`ASYMWorkWindow` becomes invalid when going to another book, even if it only reads "`viewer id 0`".

This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS No parameters.

EXAMPLES
```
ww = ASYM_WorkWindow()
```

ASYMA_AuthorIniFile()

TB89A.SBK INI Functions

SYNTAX `ASYMA_AuthorIniFile()`

DESCRIPTION Gets the name and path of the `.INI` file in which authoring preferences and information are stored.

RETURNS The full path and filename of the Authoring `INI` file `(INSTRUCTOR.INI)`.

NOTES This function is only available in the Authoring environment.

PARAMETERS No Parameters

EXAMPLES
```
-- Set the Catalog Preference to Never auto open the Catalog
get setIniVar("Options","Catalog","Never",ASYMA_AuthorIniFile())
```

at

Articles and Prepositions

DESCRIPTION A preposition used for a variety of purposes including: menu commands where the working level [`Reader` or `Author`] is being specified, when referencing a screen coordinate using the `show` or `magnify` command.

EXAMPLES

```
show linePalette at 0,0

magnify 4 at 1000,1000

add menu "Certificate" at Reader
```

atan()

Trigonometric Values

SYNTAX

```
atan(<number>)
```

DESCRIPTION Returns the arctangent of a number. The arctangent is the angle whose tangent is number. The returned angle is given in radians in the range `-pi/2` to `pi/2`.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<number>`	An expression that yields a number.

EXAMPLES

```
x = atan(1)          -- equals 0.785398 (pi/4 radians)

x = atan(1)*180/pi   -- equals 45 (degrees)
```

atan2()

Trigonometric Values

SYNTAX

```
atan2(<number1>,<number2>)
```

DESCRIPTION Returns the arctangent of the specified x- and y-coordinates. The arctangent is the angle from the x-axis to a line containing the origin (`0,0`) and a point with coordinates (number1, number2). The angle is given in radians between `-pi` and `pi`, excluding `-pi`.

PARAMETERS

PARAMETER	DESCRIPTION
`<number1>`	An expression that yields a number.
`<number2>`	An expression that yields a number.

EXAMPLES

```
x = atan2(1)          -- equals 0.785398 (pi/4 radians)

x = atan2(1)*180/pi   -- equals 45 (degrees)
```

author

Working Level

DESCRIPTION Sent to the page when you choose `Author` from the `View` menu.

You can also `send` this message using the `send author` statement. ToolBook's default response is to change the `working level` and `sysLevel` property to `Author`.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle enterPage
    SYSTEM globalErrorCatch
    if globalErrorCatch = true
        send Author
    else
        forward
    end
end
```

authorStatusBar

Property

DESCRIPTION A property of a viewer that specifies whether the `status bar` is visible at `Author` level in the viewer when it is first opened.

NOTES You can get or set this property.

VALUES `True` or `false`.

The default is `true` for the Main window, `false` for other viewers.

If `false`, the status bar is not visible when the viewer is initially displayed at `Author` level.

EXAMPLES

```
authorStatusBar of viewer "help" = false
```

autoClose

Property

DESCRIPTION A property of a viewer that specifies if a viewer is closed as the result of a single mouse click or key press at `Reader` level.

NOTES When you use the Viewer Properties dialog box to set the border style of a viewer to a shadowed frame, ToolBook automatically sets the autoClose property for that viewer to true.

VALUES `True` or `false`.

The default is `false`.

If `true` when the viewer is shown, the viewer closes when the user clicks a mouse button anywhere on the screen or presses any key.

If the user clicks an object in the viewer, that object will receive its messages before the viewer closes.

EXAMPLES

```
autoClose of viewer "Help" = true
```

autoRadioButtons

Property

DESCRIPTION A `group` property that specifies whether a group of radio buttons uses automatic selection behavior.

NOTES You can get or set this property.

This behavior will ensure that only one `radio button` is selected at a time, so when you select one `radio button` ToolBook automatically deselects the other `radio buttons` in the group.

Because this automatic behavior is available from a single property setting, you don't need to write a script to create `mutually exclusive` behavior in a group of radio buttons.

This is not a property you would typically modify at runtime, but rather is a setting normally configured while authoring your content.

VALUES `True` or `false`; the default is `true`.

If `true`, ToolBook uses the automatic selection behavior for the group of radio buttons.

EXAMPLES

```
autoRadioButtons of group "method" = true
```

autoShow

Property

DESCRIPTION A prop erty of a viewer that specifies whether a viewer is automatically opened and shown when its book is opened in the Main window.

NOTES You can get or set this property.

To make viewers that are palettes or tool bars appear whenever a book opens, set their `autoShow` property to `true`.

VALUES `True` or `false`.

The default is `false`.

EXAMPLES

```
autoShow of viewer "help" = true
```

autoSize

Property

DESCRIPTION A property of a viewer that specifies whether a viewer is automatically resized to its current page when the viewer is shown or its `currentPage` property changes.

NOTES You can get or set this property.

VALUES `True` or `false`.

The default is `true`.

If `true`, the viewer resizes itself to its current page each time the viewer is shown or its `currentPage` property changes (as a result of page navigation or if its value is set explicitly to another page).

When a viewer's `autoSize` property is set to `true`, ToolBook ignores the viewer property `defaultClientSize`.

If `false`, the viewer resizes itself to the value of the `defaultClientSize` property.

If the viewer is `maximized`, ToolBook ignores the setting for `autoSize`.

Setting `autoSize` to `true` is similar to setting `defaultClientSize` to `sizeToPage`, but using `autoSize` causes ToolBook to resize a window to the page every time a new page appears in the window rather than only when the window is first shown.

EXAMPLES

```
autoSize of viewer "Help" = true
```

average()

Statistical Values

SYNTAX `average(<list>)`

DESCRIPTION Returns the average of a list of numbers. This is done by dividing the sum of the numbers by the number of items in the list.

PARAMETERS

PARAMETER	DESCRIPTION
<list>	An expression that results in a list of numbers.

EXAMPLES

```
lst = average(14,35,66,85,940)
```

```
-- determine average character length of all page names
x = null
step k from 1 to pageCount of this book
    push charCount(name of page k) onto x
end
answ = average(x)
```

back

Navigation Message

DESCRIPTION Sent to the current page when `Back` is chosen from the `Go` menu.

You can also `send` this message using the `send back` statement. ToolBook's default response is to display the most recent page in `sysHistory` in the Main window. (The most recent page in `sysHistory` is the last page visited by the user.)

The `back` message *only* works with pages displayed in the Main window. If the target window is any viewer other than the Main window (`viewer id 0`), ToolBook's default response to this message is to do nothing.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle buttonClick
    send back
end
```

```
to handle back
    request "Sorry, you are not permitted to leave this page yet."
end
```

backdrop

Property

DESCRIPTION A background property that specifies the resource bitmap that is to be used for background tiling.

The bitmaps may have a `chromakey` in which case the background fill color and/or pattern will show through the transparent areas.

If there is no `backdrop` set, the value of backdrop is `null`.

NOTES You can get or set this property.

EXAMPLES

```
backdrop of this background = backdrop of background "army"
```

```
backdrop of backdrop "course" = bitmap "blue"
```

```
if backdrop of this background = null
    backdrop of this background = bitmap "normal"
end
```

backdropStyle

Property

DESCRIPTION A background property that describes how a backdrop is to be tiled.

NOTES You can get or set this property.

VALUES

VALUE	DESCRIPTION
`none`	No tiling at all.
`center`	Position the backdrop in the center of the background.
`stretch`	Stretch the bitmap to fill the background.
`tiled`	Tile to fill the background beginning at the upper left.
`tileCenter`	Like "`tiled`" except that top/bottom rows and left/right columns are shifted so all is centered.
`checkeredOne`	Checkerboard tiling with the upper left corner containing a bitmap.
`checkeredTwo`	Checkerboard tiling with the upper left corner showing the background fillColor and/or pattern.
`checkeredThree`	Like "`checkeredOne`" except that top/bottom rows and left/right columns are shifted so that all is centered.
`checkeredFour`	Like "`checkeredTwo`" except that top/bottom rows and left/right columns are shifted so that all is centered.

EXAMPLES

```
backdropStyle of this background = "tileCenter"

backdropStyle of this background = backdropStyle of background "red"

if backdropStyle of this background = "none"
    backdrop of this background = bitmap "center"
end
```

background

Working Layer

DESCRIPTION Sent to the current page when you choose `Background` from the `View` menu.

You can also `send` this message using the `send background` statement. ToolBook's default response is to switch from the page to the background of the current page.

You can use the `background` message to select objects on the background from a script. Alternately, you can set the `onBackground` property of the viewer showing the background to `true`.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
-- Adds a recordfield to a book
to handle addRecordField pLoc1, pLoc2, pName
    sysLockScreen = true
    send background
    draw recordfield from pLoc1 to pLoc2
    name of selection = pName
    send foreground
    sysLockScreen = false
end
```

background

Object Type

DESCRIPTION The object type name for a `background`, which is a template shared by pages in a book.

NOTES Every book has at least one background, and all of its objects appear in the same position, style, and size on every `page` sharing that background.

The background's parent is its book. A background is the parent of a page.

To create a background, use the `send newBackground` statement.

TO REFER TO	USE THIS SYNTAX
Currently displayed background	`this background` `parent of this page`
Background in the current book	`background "Trees"` `background ID 4`
Background in another book	`background "Trees" of book "land.tbk"`

Each background has a `size` property that defines the size of its pages in `page units`. Changing this property changes the display size and scrolling extent of a background's pages. If the value of a background's size property is `0,0`, the background inherits the book's `size`. The `size` property for new backgrounds is `0,0` by default. Each background in a book can be a different size.

The `activated` property of `fields` on the background is always `true`, so users cannot type text into the fields on backgrounds at `Reader` level. `Recordfields` can be placed only on backgrounds, although the text of a recordfield is unique to each page that shares the background.

The `enterBackground` and `leaveBackground` messages are sent to the current page when the user navigates between pages with different backgrounds.

The `destroy` and `make` notification messages are sent to the background when it is created or deleted.

Place a handler in a background script to define a general behavior for pages that share the background, or for objects on those pages and the background.

To select an object on the background, send the `background` message before selecting the object or set the `onBackground` property of a viewer to `true` to switch to the background of a page displayed in the viewer.

To conceal switching between the foreground and background, set `sysLockScreen` to `true` first.

backgroundCount

Property

DESCRIPTION A book property that indicated the number of backgrounds in a book.

NOTES This is a read only property and cannot be set.

`BackgroundCount` is equal to `itemCount(backgrounds of this book)`.

VALUES A positive whole number indicating the number of backgrounds in a book.

EXAMPLES

```
-- Does the book contain a background named "Course"?
found = false
step k from 1 to backgroundCount of this book
    if name of item k of backgrounds of this book = "Course"
        found = true
        break step
    end
end
```

backgrounds

Property

DESCRIPTION A book property that holds a list of all the backgrounds in a book.

NOTES This is a read only property and cannot be set.

VALUES A comma separated list of background references.

EXAMPLES

```
-- Does the book contain a background named "Course"?
found = false
step k from 1 to backgroundCount of this book
    if name of item k of backgrounds of this book = "Course"
        found = true
        break step
    end
end
```

baselines

Property

DESCRIPTION A field or recordfield property that specifies whether text `baselines` in a field or recordfield are visible.

NOTES You can get or set this property.

Baselines indicate visually where the base of each line of text is, in the same way that a sheet of college ruled paper denotes the base of each line.

VALUES `True` or `false`; the default is `false`.

EXAMPLES

```
baselines of field "notepad" = true
```

beep

Sound Commands

SYNTAX

```
beep <number>
```

DESCRIPTION Plays the default `beep` sound defined in the `Windows Control Panel`.

NOTES If the default beep cannot be played (usually because a sound card or sound driver isn't present), Windows generates a beep using the computer's speaker.

PARAMETERS

PARAMETER	DESCRIPTION
`<number>`	A positive integer that specifies how many times to beep.

EXAMPLES

```
beep 2
```

before

Articles and Prepositions

DESCRIPTION Used with various OpenScript commands [such as pop, push, insert graphic] to indicate that a value should precede other values.

EXAMPLES

```
put "The End" before text of field "story"
```

```
insert graphic bitmap "Fog" before text of focus
```

```
pop mylist before goodList
```

bitAnd

Bit Operators

SYNTAX `<expression> bitAnd <expression>`

DESCRIPTION Performs a bitwise `AND` operation on two numbers.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that results in a numeric value.

EXAMPLES

```
request 3 bitAnd 2
request 0x11 bitAnd 0x10
```

bitmap

Resource Type

DESCRIPTION The resource type name for a `bitmap` resource.

Bitmap resources are referenced by `name` or by `ID` with the `bitmap` keyword. The `ID` is assigned by the ToolBook system, but you can assign any `name` to the resource.

You can set a bitmap resource for the `checkedGraphic`, `disabledGraphic`, `invertGraphic`, and `normalGraphic` properties of a graphic button, and for the `dragImage` and `noDropImage` object properties.

EXAMPLES

```
normalGraphic of button "exit" = bitmap "downArrow"
```

bitNot

Bit Operators

SYNTAX `bitNot <expression>`

DESCRIPTION Converts all bits to their opposite value by performing a one's complement of a number.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that results in a numeric value.

EXAMPLES

```
request bitNot 6
```

bitOr

Bit Operators

SYNTAX `<expression> bitOr <expression>`

DESCRIPTION Performs a bitwise inclusive `OR` operation on two numbers.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that results in a numeric value.

EXAMPLES

```
request 3 bitOr 2
request 0x11 bitOr 0x10
```

bitShiftLeft

Bit Operators

SYNTAX `<expression> bitShiftLeft <amount>`

DESCRIPTION Shifts a number's bits to the left by a specified amount.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that results in a numeric value.
`<amount>`	A number representing the number bits to shift to the left.

EXAMPLES

```
request 2 bitShiftLeft 1      -- 4
request 2 bitShiftLeft 2      -- 8
```

bitShiftRight

Bit Operators

SYNTAX `<expression> bitShiftRight <amount>`

DESCRIPTION Shifts a number's bits to the right by a specified amount.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that results in a numeric value.
`<amount>`	A number representing the number bits to shift to the right.

EXAMPLES

```
request 4 bitShiftRight 1      -- 2
request 4 bitShiftRight 2      -- 1
```

bitXOr

Bit Operators

SYNTAX `<expression> bitXOr <expression>`

DESCRIPTION Performs a bitwise `exclusive OR` operation on two numbers.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that results in a numeric value.

EXAMPLES

```
request 3 bitXOr 2
```

black

Color Constants

DESCRIPTION Represents the color `black`. This is equivalent to the `RGB` value of `0,0,0` and the `HLS` value of `0,0,0`.

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES

```
rgbFill of field "list" = black
```

```
if rgbStroke of rectangle "drop area" = black
    rgbStroke of rectangle "drop area" = 0,191,0
end
```

```
strokeColor of ellipse id 14 = black
```

blue

Color Constants

DESCRIPTION Represents the color `blue`. This is equivalent to the `RGB` value of `0,0,255` and the `HLS` value of `240,50,100`.

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES

```
rgbFill of field "list" = blue
```

```
if rgbStroke of rectangle "drop area" = blue
    rgbStroke of rectangle "drop area" = 0,191,0
end
```

```
strokeColor of ellipse id 14 = blue
```

bold

Character Formatting

DESCRIPTION Sent to the current page when `Bold` is chosen from the `Text` menu.

You can also `send` this message using the `send bold` statement. ToolBook's default response is to change the type style to bold or, if it's already bold, to turn off that style.

EXAMPLES

```
select chars 15 to 20 of text of field "list"
send bold
```

book

Object Type

DESCRIPTION The object type name for a `book`. A `book` is the basic unit of a ToolBook application.

NOTES Books do not have parents, but the book is the parent of all `backgrounds` and all `viewers`. System books are higher than books in the ToolBook object hierarchy.

To create a new book, use the `send new` statement.

To save a book, use the `send save` statement.

TO REFER TO	USE THIS SYNTAX
Active book	`this book` `parent of this background`
Book in the current directory	`book "landscape.tbk"`
Book in a different directory	`book "c:\home\land.tbk"`

The `enterBook` and `leaveBook` messages are sent to the current page. There are no built-in messages that are sent directly to the book, but messages can travel up the hierarchy or be sent from scripts to the book.

Place a handler in a book's script to define general behavior for certain objects in a book, such as a button that creates navigation to the next page. Also, place handlers in a book's script for user functions, messages, and properties that are used throughout the book.

To display a page from another book without opening that book's instance, set the `currentPage` property of a viewer to a valid page reference in another book. The page is displayed in the viewer.

bookURL

UNDOCUMENTED Property

DESCRIPTION When running in Neuron, this property returns the URL from which the book was downloaded.

When running without Neuron, this property returns the `name` of the book.

NOTES When a book is run in Neuron, the `name` of the book is the file in browser cache containing the local copy.

WARNING THIS IS AN UNDOCUMENTED FEATURE. What that means is that you can use it but you will most likely be unable to get assistance from Click2learn on how to use it, or troubleshoot it.

VALUES A string in the form of either a URL or a local filename.

EXAMPLES

```
to handle buttonClick
    request name of this book && "was downloaded from" && \
      bookURL of this book
end
```

bookVersion()

File Function (General)

SYNTAX

```
bookVersion(<file name>)
```

DESCRIPTION Determines which version of ToolBook last saved a ToolBook book.

RETURNS Possible values returned by this function are `1.0`, `1.5`, `3.0`, `3.0MM`, `4.0`, `5.0`, `6.0`, `7.0`, `7.2`, `8.0` or `8.5`.

If the specified file is not recognized as a ToolBook book, the function returns `null`.

NOTES Call this function to determine the file format of a .TBK file without opening the file, which prevents the book from being converted to the version of ToolBook currently running.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	The name of a ToolBook book file, including the path if necessary.

EXAMPLES

```
if bookVersion("index15.tbk") <> bookVersion(name of this book)
    request "Your files are out of date, please reinstall the " & \
            "current version of this software."
end
```

borderStyle

Property

DESCRIPTION A combobox, button, field, recordfield, or viewer property that specifies a border style.

NOTES You can get or set this property.

VALUES For buttons: `checkBox`, `checkBox3D`, `commandButton`, `label`, `none`, `pushButton`, `radioButton`, `radioButton3D`, `rectangle`, `rounded`, or `shadowed`; the default is `pushButton`.

Set a button's `borderStyle` to `label` to create a static label that contains read-only, textual information. When a button that uses the `label` border style gets the focus, the focus is passed to the next editable object in the tabbing order.

For fields and recordfields: `inset`, `none`, `raised`, `rectangle`, `scrolling`, or `shadowed`; the default is `rectangle`.

For viewers: `dialogFrame`, `none`, `shadowed`, `thickFrame`, or `thinFrame`; the default is `thickFrame`. When a viewer's borderStyle property is set to `dialogFrame`, the viewer's `state` is always `normal` (ToolBook ignores the settings for `defaultState` and `state`), and only the `Move` and `Close` commands are available in the viewer's Control menu box.

If a viewer's `borderStyle` is set to `shadowed`, the viewer is displayed with a shadow that is offset below and to the right of the actual viewer. If the `autoClose` property of the shadowed viewer is `true` when the viewer is shown, the shadowed viewer automatically closes when the user clicks the mouse button once or presses a key.

EXAMPLES

```
borderStyle of button "start" = "shadowed"
```

borderWidth

Property

DESCRIPTION A stage property that specifies the width of the stage frame border.

NOTES You can get or set this property.

VALUES A number in `page units`.

The default is `45`.

EXAMPLES

```
borderWidth of stage "player" = 90
```

bounds

Property

DESCRIPTION A property of almost all ToolBook object types that specifies the current `size` and `position` of an object's bounding rectangle.

NOTES You can get or set this property.

For ToolBook `draw objects`, the bounds are represented in `page units`.

For viewers, the `Command Window`, and `palettes`, the bounds are represented in `pixels`.

For buttons, comboboxes, ellipses, fields, recordfields, OLE objects, paint objects, pictures, rectangles, rounded rectangles, stages, and viewers, the bounds and `vertices` of the object are the same. The bounds of a `combobox` include its edit field and pushbutton, but not its drop-down list box.

For `arcs`, `angled lines`, `curves`, `irregular polygons`, or `lines` that slope downward from left to right, the bounds are the same as the location of the upper-left and lower-right corners of the object's selection handles.

For `lines` without line ends, that are perfectly vertical, horizontal, or that slope downward from left to right, the `bounds` are the same as the `vertices`. For lines with line ends, the bounds refer to the smallest rectangle that encompasses both the line and its line ends; the `bounds` for these lines change to reflect the current setting for the line's `lineEndStyle` and `lineEndSize` settings.

For `hotwords`, the bounds refer to the smallest rectangle that encompasses the hotword's text. If the hotword's text spans several lines, the left and right edges of the bounding rectangle align with the left and right edges of the field. If the hotword is scrolled out of view, getting the bounds returns 0,0,0,0.

For `viewers`, the bounds of a child window are relative to the client area of the parent window; the bounds of a popup window are relative to the desktop.

VALUES A list of four numbers, where the first two numbers specify the `x,y` coordinates of the upper-left corner of the bounding rectangle, and the last two numbers specify the `x,y` coordinates of the lower-right corner.

For ToolBook draw objects, the numbers are in `page units`.

For viewers, the Command window, and palettes, the numbers are in `pixels`.

EXAMPLES

```
-- set the size and position
bounds of rectangle "Apple" = 10,10,400,800

-- ensure the bottom of this object is not exceeding limit
if item 4 of bounds of ellipse "red" > 4000
    item 4 of bounds of ellipse "red" = 4000
end
```

break

Program Flow

SYNTAX

```
break
break <handler name>
break <control structure>
break to system
```

DESCRIPTION Causes a break in the execution of a script and jumps to the end of the current handler or control structure.

The handler ends in the normal way by passing control back to the calling handler. The `break to system` form bypasses any other handlers and passes control directly to ToolBook.

NOTES Within a to get handler, you can use the `break to system` and `break <control structure>` forms, but you cannot use the `break` or `break <handler>` forms. Control can be passed from a `to get` handler to a calling handler only with a `return` statement.

Using the `break` command within an `if/then/else` control structure causes the execution of the script to jump to the end of the currently executing handler, not to the end of the control structure.

Use a `break to system` statement within a `notifyAfter` or `notifyBefore` handler to stop the handler from processing a message in its script.

PARAMETERS

PARAMETER	DESCRIPTION
`<handler name>`	The name of the handler from which to exit.
`<control structure>`	The name of the control structure: `conditions`, `do`, `step`, or `while`.

EXAMPLES

```
step k from 1 to 100
    increment ctr by k
    if ctr > 170
        break step k
    end
end

while var <> null
    pop var into curVal
    send processInfo(curVal)
    if keyState(keyEscape) = "down"
        break to system
    end
end

to handle buttonClick
    SYSTEM ttl
    increment ttl
    if ttl > 100
        request "Overload has occurred, abort now?" with "Yes" or "No"
        if IT = "Yes"
            break buttonClick
        end
    end
    get processMold(ttl)
end
```

buildCacheFile

Property

DESCRIPTION A book property that specifies whether a CD-ROM application uses a cache file.

NOTES You can get or set this property.

Use this property if you are writing a script that saves a book as optimized for CD-ROM. You can also set the `buildCacheFile`, `cacheFileType`, and `sysOptimizedSave` properties using the `Save As` or `Save As EXE` dialog box.

To significantly improve startup speed in a large CD-ROM application that uses a cache file, set `buildCacheFile` to `permanent` rather than `temporary`.

VALUES `Never`, `temporary`, or `permanent`.

The default is `never`.

If `never`, no cache file is used.

If `temporary`, ToolBook copies the optimized data from the CD-ROM to the hard disk each time the application opens, and deletes the data from the hard disk when it closes.

If `permanent`, the Setup Program should copy a permanent cache file to the user's hard disk when the application is installed; ToolBook uses this cache file whenever the application runs.

When a user installs an application that uses a permanent cache file, the Setup Program should record the cache file location in the `TB89.INI` file under `[CACHE FILES]`. The information stored in the .INI file includes the name of the book (without its path) set to the full path of the cache file (*.TBC).

For example:

```
[CACHE FILES]
TOUR.TBK = C:\BOOKS\CACHE\TOUR.TBC
```

EXAMPLES

```
-- save this book with a cache file
buildCachFile of this book = permanent
cacheFileType of this book = preferred
save as "c:\books\new.tbk", true
```

button

Object Type

SYNTAX `draw button from <location> to <location>`

DESCRIPTION The object type name for `buttons`.

NOTES A button's parent can be the page, the background, or a group.

TO REFER TO A BUTTON	USE THIS SYNTAX
On the current page	`button "Quit"` `button ID 27`
In a group	`button "next" of group "controls"`
On another page	`button "Star" of page 7`
On another background	`button "Stop" of background "DataForm"`
In another book	`button "x1" of page "L6" of book "B1.tbk"`

A button's enabled property specifies whether the button can receive the input focus or mouse event messages at `Reader` level. When a button's enabled property is set to false, the button is disabled; it cannot receive keyboard input or mouse event messages and is excluded from the tabbing order.

The `enterButton` and `leaveButton` messages are sent automatically to a button at `Reader` level when it gets or loses the focus. All the mouse event and keyboard event messages are sent automatically to buttons at `Reader` level. The `destroy`, `make`, `moved`, and `sized` messages are also sent automatically to buttons.

ToolBook automatically toggles the value of the checked property of a `radio button` or a `check box`

every time a user clicks the button.

The `autoRadioButtons` group property, when `true`, causes a group of `radio buttons` to behave in a `mutually exclusive` manner when selected. `Radio buttons` must be grouped to use this property, which is `true` by default.

To make a button look as if it's pushed, set its `invert` property to `true`. If you don't want a button to get the focus (so that the focus outline doesn't appear, or to prevent keyboard execution of scripts), set `excludeTab` to `true`. If the button's `caption` is `null` and it does not use a graphic, the focus outline never appears in the button, but if `excludeTab` is `false` the user can press Tab to move the focus to it.

buttonClick

Event Message

SYNTAX

```
buttonClick <location>,<isShift>,<isCtrl>
```

DESCRIPTION

Sent to the object that the mouse cursor is pointing at when the left mouse button is pressed and released.

NOTES

If the user drags the cursor outside the bounds of the target object before releasing the mouse button, the `buttonClick` message is not sent.

ToolBook supplies three parameters that specify the location of the cursor when the mouse button is pressed, and whether the `Shift` and/or `Ctrl` key is also pressed.

When the user clicks an object, the object receives these messages in the following order: `buttonDown`, `buttonUp`, `buttonClick`.

Use a `buttonClick` message handler in place of a `buttonUp` handler whenever the task does not specifically require a `buttonUp` message.

EXTRA NOTE

As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
`<location>`	A list of two numbers in `page units`, indicating the location of the cursor when the mouse button is pressed.
`<isShift>`	`True` or `false`, indicating whether the `Shift` key is pressed in conjunction with the mouse button.
`<isCtrl>`	`True` or `false`, indicating whether the `Ctrl` key is pressed in conjunction with the mouse button.

EXAMPLES

```
to handle buttonClick
    if target is button "More info"
        go to page "history" of book "facts.tbk"
    end
    forward
end
```

```
to handle buttonClick loc, sft, ctrl
    if ctrl = true
        start spooler
            print
        end
    end
    forward
end
```

buttonDoubleClick

Event Message

SYNTAX `buttonDoubleClick <location>,<isShift>,<isCtrl>`

DESCRIPTION Sent to the object that the mouse cursor is pointing at when the left mouse button is clicked twice within the double-click time specified in the Windows Control Panel.

NOTES ToolBook supplies three parameters that specify the location of the cursor when the mouse button is pressed, and whether the `Shift` and/or `Ctrl` key is also pressed.

When the user double-clicks an object, the object receives these messages in the following order: `buttonDown, buttonUp, buttonClick, buttonDoubleClick, buttonUp.`

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
`<location>`	A list of two numbers in `page units`, indicating the location of the cursor when the mouse button is pressed.
`<isShift>`	`True` or `false`, indicating whether the `Shift` key is pressed in conjunction with the mouse button.
`<isCtrl>`	`True` or `false`, indicating whether the `Ctrl` key is pressed in conjunction with the mouse button.

EXAMPLES

```
to handle buttonDoubleClick
    if target is button "More info"
        go to page "history" of book "facts.tbk"
    end
    forward
end

to handle buttonDoubleClick loc, sft, ctrl
    if ctrl = true
        start spooler
            print
        end
    end
    forward
end
```

buttonDown

Event Message

SYNTAX `buttonDown <location>,<isShift>,<isCtrl>`

DESCRIPTION Sent to the object that the mouse cursor is point ing at when the left mouse button is pressed and held.

NOTES ToolBook supplies three parameters that specify the location of the cursor when the mouse button is pressed, and whether the `Shift` and/or `Ctrl` key is also pressed.

When the user clicks an object, the object receives these messages in the following order: `buttonDown`, `buttonUp`, `buttonClick`.

If a `button` or `hotword` is the target of a `buttonDown` message and its `highlight` property is `true`, the `button` or `hotword` is `highlighted`. A `graphic` button displays its `invert graphic` before the `buttonDown` message is sent up the hierarchy.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
<location>	A list of two numbers in page units, indicating the location of the cursor when the mouse button is pressed.
<isShift>	True or false, indicating whether the Shift key is pressed in conjunction with the mouse button.
<isCtrl>	True or false, indicating whether the Ctrl key is pressed in conjunction with the mouse button.

EXAMPLES

```
to handle buttonDown
    if target is button "More info"
        go to page "history" of book "facts.tbk"
    end
    forward
end

to handle buttonDown loc, sft, ctrl
    if ctrl = true
        start spooler
            print
        end
    end
    forward
end
```

buttonStillDown

Event Message

SYNTAX

```
buttonStillDown <location>,<isShift>,<isCtrl>,<isRight>
```

DESCRIPTION

Sent at Reader level to the object that contains the cursor in its active area when the left or right mouse button is pressed. ToolBook sends the buttonStillDown message repeatedly as long as the mouse button remains pressed, and continuously updates the <location> parameter to identify the cursor's current location.

NOTES

Use the <isRight> parameter to find out which mouse button is pressed.

The buttonStillDown message is useful for applications that need to track mouse drag movement, such as in scrolling, layout control, or animation.

EXTRA NOTE

As with most all system generated messages, it is important to forward the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
<location>	A list of two numbers in page units, indicating the location of the cursor when the mouse button is pressed.
<isShift>	True or false, indicating whether the Shift key is pressed in conjunction with the mouse button.
<isCtrl>	True or false, indicating whether the Ctrl key is pressed in conjunction with the mouse button.
<isRight>	True or false. If true, the right button is still down; if false, the left button is still down.

EXAMPLES

```
--Draws a rectangle with dotted outline at Reader level
to handle buttonDown pLoc
    draw rectangle from pLoc to pLoc
    name of selection = "outline"
    lineStyle of selection = dotted
    forward
end

to handle buttonStillDown pLoc
    items 3 to 4 of bounds of rectangle "outline" = pLoc
    forward
end

to handle buttonUp
    lineStyle of rectangle "outline" = 1
    forward
end
```

buttonUp

Event Message

SYNTAX

`buttonUp <location>,<isShift>,<isCtrl>`

DESCRIPTION

Sent at `Reader` level when the left mouse button is released.

NOTES

This message is sent to the object that contained the cursor in its active area when the left mouse button was pressed, causing the `buttonDown` message to be sent.

ToolBook supplies three parameters that specify the location of the cursor when the mouse button was pressed, and whether the `Shift` or `Ctrl` key is also pressed.

When the user clicks an object, the object receives these messages in the following order: `buttonDown`, `buttonUp`, `buttonClick`.

If a `button` or `hotword` is the target of a `buttonUp` message and its `highlight` property is `true`, the button's or hotword's `highlighting` is turned off before the `buttonUp` message is sent. A `graphic button` displays its `invert graphic` before the `buttonDown` message is sent up the object hierarchy.

EXTRA NOTE

As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
`<location>`	A list of two numbers in `page units`, indicating the location of the cursor when the mouse button is pressed.
`<isShift>`	`True` or `false`, indicating whether the `Shift` key is pressed in conjunction with the mouse button.
`<isCtrl>`	`True` or `false`, indicating whether the `Ctrl` key is pressed in conjunction with the mouse button.

EXAMPLES

```
to handle buttonUp
    if target is button "More info"
        go to page "history" of book "facts.tbk"
    end
    forward
end

to handle buttonUp loc, sft, ctrl
    if ctrl = true
        start spooler
            print
        end
    end
    forward
end
```

by

Articles and Prepositions

DESCRIPTION Used in conjunction with various OpenScript terms to indicate amounts. This includes the following: `decrement`, `increment`, `move`, `search`, `sort` and `step`.

EXAMPLES

```
decrement x by 3

increment x by 10

move rectangle "t14" by 20,30

step k from 100 to 1 by -1
```

cacheFileType

Property

DESCRIPTION A book property that specifies the amount of data saved to a cache file used by a CD-ROM application.

NOTES You can get or set this property.

Use this property if you are writing a script that saves a book as optimized for CD-ROM.

You can also set the `buildCacheFile`, `cacheFileType`, and `sysOptimizedSave` properties using the `Save As` or `Save As EXE` dialog box.

VALUES `Minimal`, `preferred`, or `extended`.

The default is `minimal`.

If `minimal`, only basic page information is saved to the cache file. This setting requires the least disk space of the three.

If `preferred`, page information and references to resources are saved to the cache file. This setting usually provides the best results, but results vary depending on the application.

If `extended`, page information and the actual resource data are saved to the cache file. This setting requires the most disk space.

EXAMPLES

```
-- savet this book with a cache file
buildCacheFile of this book = permanent
cacheFileType of this book = preferred
save as "c:\books\new.tbk", true
```

caption

Property

DESCRIPTION A button, book or a viewer property that specifies the text to be shown as the object's title.

A status bar property that specifies the text displayed in the status bar.

NOTES You can get or set this property.

Setting the `caption` for a book has the same effect as setting the `caption` property of `mainWindow` or `viewer id 0`.

To display custom text in the status bar, set the `caption` property of `statusBar` to a string of text.

VALUES For `viewers`, a string of up to 78 characters; the default is `null`.

For `buttons`, a string of up to 255 characters; the default is `&Button`. The `ampersand` (`&`) makes a character in the button's caption a `mnemonic` access character.

If the `caption` property of a button without a graphic is `null`, the `focus` outline does not appear in the button.

If caption is `null` on a button with a graphic, the `focus` outline appears around the graphic. A user can press Tab to move the `focus` to that button if `excludeTab` is `false`.

For the `statusbar`, a string of up to 255 characters; the default is `null`.

ACTIONS EDITOR The Actions Editor also supports this feature for Button objects.

EXAMPLES

```
caption of this window = "An American History"
caption of button "start" = "&Start Now"
caption of statusBar = "Click this button to begin"
```

captionBar

Property

DESCRIPTION A property of a viewer that specifies its type of caption bar.

NOTES You can get or set this property.

VALUES `None`, `normal`, or `thin`.

The default is `normal`.

EXAMPLES

```
captionBar of mainWindow = "thin"
captionBar of viewer "Help" = "none"
```

captionPosition

Property

DESCRIPTION A button property that specifies the position of a button's caption relative to its graphic.

NOTES You can get or set this property.

VALUES `Auto`, `bottom`, `center`, `left`, `right`, or `top`.

The default is `auto`.

When a button's `captionPosition` is set to `auto`, and a button that isn't a `radio button` or `check box` uses a graphic, the `caption` is displayed below the graphic.

If the button does not have a graphic, the `caption` is centered.

For `check boxes` and `radio buttons` that use graphics, the `caption` is displayed to the right of the graphic.

EXAMPLES

```
captionPosition of button "lighting" = "bottom"
```

caretFontFace

Property

DESCRIPTION A viewer property that specifies the current font face for the viewer.

NOTES You can get or set this property.

This property determines the font face to be used the next time a character is typed in a field or recordfield in the viewer.

If a viewer is not specified, the target viewer is assumed.

VALUES The value of the property is the name of the font to be used at the caret location.

EXAMPLES
```
caretFontFace of viewer "form" = "Arial"
```

caretFontSize

Property

DESCRIPTION A viewer property that specifies the current font size for the viewer.

NOTES This property determines the font size to be used the next time a character is typed in a field or recordfield in the viewer.

If a viewer is not specified, the target viewer is assumed.

VALUES A positive whole number that defines the point size of the font face.

EXAMPLES
```
caretFontSize of viewer "form" = 14"
```

caretFontStyle

Property

DESCRIPTION A viewer property that specifies the current font style for the viewer.

NOTES You can get or set this property.

This property determines the font style to be used the next time a character is typed in a field or recordfield in the viewer.

If a viewer is not specified, the target viewer is assumed.

VALUES `Regular` (or `null`) or a list of one or more of the values: `bold`, `italic`, `underline`, `strikeout`, `superscript`, or `subscript`.

EXAMPLES
```
caretFontStyle of viewer "form" = "underline,bold"
```

caretLocation

Property

DESCRIPTION Specifies the insertion point (I-Beam) for text within a field or recordfield.

NOTES You can get or set this property.

The insertion point appears in the space between characters, not in the location of a character itself.

Note that `caretLocation` was formerly a `system property` and will still behave as if it were a system property if no viewer is specified when referring to the property. In such a case, the target viewer is assumed.

VALUES A list of two positive whole numbers, `home`, or `end`.

The first integer specifies the number of the textline where the caret is located.

The second integer specifies the number of characters between the beginning of the text line and the caret, a number that lies between 0 and the total number of characters in the line of text.

Use the `home` or `end` keyword in a statement with `caretLocation` to move the insertion point to the beginning or end of the current field or recordfield.

EXAMPLES
```
caretLocation of viewer "form" = 2,15
caretLocation of viewer "help" = "end"
```

CDMediaPath

Property

DESCRIPTION A book property that specifies the CD-ROM directories in which ToolBook searches for external media referenced by `clips`.

NOTES You can get or set this property.

ToolBook uses the value of this property when a clip's `mmSearchCD` property is `true`.

Use this property to avoid path dependencies in your scripts and simplify the organization and delivery of clips in your application.

VALUES A comma separated list of directories.

Add the string value `<BookPath>` to indicate the current book's path.

For best performance, restrict the CD-ROM path to only one directory.

EXAMPLES

```
-- set CDMediaPath to the current CD Drive letter and sub path
driveLetter = fetchTheCDRomDriveLetter()
baseCDPath = ":\allmedia\"
CDMediaPath = driveLetter & baseCDPath
```

ceiling()

Arithmetic Values

SYNTAX `ceiling(<number>)`

DESCRIPTION Rounds a number up to the nearest whole number.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<number>`	An expression that results in a number.

EXAMPLES

```
val = 4.56
x = ceiling(val) -- results in x being set to 5

val = -4.56
x = ceiling(val) -- results in x being set to -4
```

centerClient

Property

DESCRIPTION A property of a viewer that specifies whether a viewer's client window is centered within the frame of the viewer when the viewer is larger than necessary to fully display the page.

NOTES You can get or set this property.

The client area of a viewer is the entire area contained within the viewer's frame.

The client window of the viewer is the area that displays a page.

The viewer's client area can be larger than the client window.

VALUES `True` or `false`.

The default is `true`.

If `true`, the viewer's client window is centered within the viewer's frame.

If `false`, the client window is positioned in the upper-left corner of the viewer's client area.

EXAMPLES

```
centerClient of viewer "help" = true
```

char, chars, character, characters

String Operators

SYNTAX

```
char <number> of <stringRef>
character <number> of <stringRef>
chars <number> to <number> of <stringRef>
characters <number> to <number> of <stringRef>
```

DESCRIPTION

A string operator used to indicate one or more characters of text.

NOTES

See the last example to find out how you can replace a single character with many characters using this operator. Using this technique you can replace any number of characters in a string with any number of other characters.

ACTIONS EDITOR

The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<number>	An expression that results in a number.
<stringRef>	An expression that results in a string value.

EXAMPLES

```
-- if part number is type H remove from list
if char 4 of textline k of text of field "part number" = "H"
    clear textline k of text of field "part number"
end
```

```
- Flip Slashes
if chars 1 to 7 of str = "http:\\"
    chars 6 to 7 of str = "//"
end
```

```
-- Clear last character if it is an A, B or C
lst = "ABC"
step k from 1 to charCount(lst)
    if last character of str = char k of lst
        clear last char of str
    end
end
```

```
-- Replace percent token character with full word
if first char of val = "%"
    first char of val = "Percent"
end
```

charCount()

String Functions

SYNTAX

```
charCount(<expression>)
```

DESCRIPTION

Counts the number of characters in an expression.

RETURNS

Returns the number of characters in a string.

Note that a carriage return is typically represented in ToolBook as `CRLF`, and `CRLF` counts as two characters.

If the expression contains no characters then `0` will be returned.

NOTES

You can also count words with `wordCount()` and textlines [textline essentially means paragraph] with `textlineCount()`.

ACTIONS EDITOR

The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<expression>	An expression that results in a string value.

EXAMPLES

```
-- Check to see if Poem is long enough
to handle buttonClick
    ttl = charCount(text of field "poem")
    if ttl < 1000
        request "Sorry, you need at least 1000 characters."
    end
end

-- incorporate a simple encryption scheme
to handle buttonClick
    txt = text of field "secure"
    step k from 1 to charCount(txt)
        curChar = char k of txt
        ansiVal = charToAnsi(curChar)
        ansiVal = ansiToChar((ansiVal + 1) mod 255)
        char k of txt = ansiVal
    end
    text of field "secure" = txt
end
```

charToAnsi()

String Functions

SYNTAX

```
charToAnsi(<character>)
```

DESCRIPTION

Converts a `ANSI` numeric value into a character.

RETURNS

Returns a number between `0` and `255`, representing the `ANSI` value of the character.

NOTES

You can also convert an `ANSI` value back into a character by using the `ansiToChar()` function.

PARAMETERS

PARAMETER	DESCRIPTION
`<character>`	A single character. If you pass more than a single character, only the first character will be converted.

EXAMPLES

```
-- incorporate a simple encryption scheme by incrementing each ANSI
-- value by 1. Example: A becomes B, 8 becomes 9, etc.
to handle buttonClick
    txt = text of field "secure"
    step k from 1 to charCount(txt)
        curChar = char k of txt
        ansiVal = charToAnsi(curChar)
        ansiVal = ansiToChar((ansiVal + 1) mod 255)
        char k of txt = ansiVal
    end
    text of field "secure" = txt
end
```

check menuItem

Menu Command/Function

SYNTAX

```
check menuItem <name | alias> [in <menu reference> [in <menu
  reference>] ... ] [at <level>]
```

DESCRIPTION

Puts a `checkmark` to the left of a `menu item` on the menu bar of the target window.

To remove a checkmark from a menu item, use the `uncheck menuItem` command.

NOTES

You can add a `checkmark` to a `menu item` in a submenu using the in `<menu reference>` parameter.

PARAMETERS

PARAMETER	DESCRIPTION
`<name | alias>`	The name of the menu item as it appears on the menu, or the alias assigned to the menu item.
`<menu reference>`	A valid reference to an existing menu or submenu in the target window. If more than one menu shares the same reference, ToolBook acts on the first menu from the left on the menu bar.
`<level>`	`Author`, `Reader`, or `both`. If a level is not specified, the current working level is assumed.

EXAMPLES

```
check menuItem "Book List" in menu "Options" at Reader
```

```
--Sets up a menu item that can be toggled
to handle soundEffects
    if not menuItemChecked("Sound Effects")
        check menuItem "Sound Effects"
        send startSoundEffects
    else
        uncheck menuItem "Sound Effects"
        send stopSoundEffects
    end
end
```

checked

Property

DESCRIPTION A button property that specifies whether a check box is checked or a radio button is selected.

NOTES You can get or set this property.

Setting the `checked` property of a button with a `commandButton` border style draws a black rectangle around the button border.

VALUES `True` or `false`.

The default is `false`.

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES

```
-- make sure R1 and R2 are not checked at the same time.
checked of button "R1" = not checked of button "R2"
```

```
-- do we need to print?
if check of button "print" = true
    go to page "print screen"
end
```

checkedGraphic

Property

DESCRIPTION A button property that specifies a graphic resource assigned to be the resource shown when the button's `checked` property is set to `true`.

NOTES You can get or set this property.

The `checkedGraphic` property is used specifically when a button's `borderStyle` property is `checkBox`, `checkBox3D`, `radioButton`, or `radioButton3D`.

VALUES A valid resource reference.

You can assign an `icon`, `cursor`, or `bitmap resource`.

If `null`, ToolBook displays the image that is set for the `normal graphic`.

EXAMPLES

```
checkedGraphic of button "Print" = icon "printer"
```

chooseColorDlg()

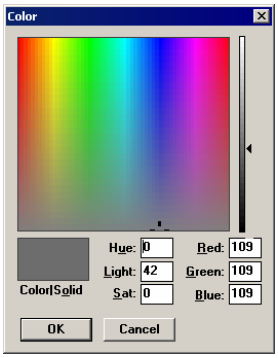
TBDLG.DLL Color Functions

SYNTAX

```
chooseColorDlg(<red value>,<green value>,<blue value>)
```

DESCRIPTION

Displays the standard Windows Color dialog box used for creating custom colors in the `Color Tray` at `Author` level in ToolBook.



RETURNS

If no error occurs, `chooseColorDlg()` returns the `RGB` value of the color chosen by the user. Otherwise, the function returns `null` and `sysError` is set to one of these values:

`-1` User canceled the dialog box.
`-2` Memory allocation error.

To assign a color returned from the `chooseColorDlg()` function to an object, use the `rgbStroke` and `rgbFill` properties.

NOTES

As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING chooseColorDlg(BYTE,BYTE,BYTE)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<red value>`	Red component of the default `RGB` color (a value between `0` and `255`)
`<green value>`	Green component of the default `RGB` color (a value between `0` and `255`)
`<blue value>`	Blue component of the default `RGB` color (a value between `0` and `255`)

EXAMPLES

```
get chooseColorDlg(255,255,255)
rgbFill of rectangle "x2" = IT
```

chooseDirectoryDlg()

TBDLG.DLL File Function (Dialog)

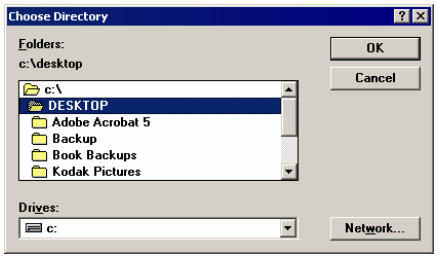
SYNTAX

```
chooseDirectoryDlg(<caption text>,<default path>)
```

DESCRIPTION

Displays the Choose Directory dialog box (based on the standard Windows Open File dialog box).

This function will accept a long file name as input, but will return short file names only. Use `chooseDirectoryDlgLFN()` if you need to return long file names.



RETURNS

If no error occurs, `chooseDirectoryDlg()` returns a string containing the directory chosen by the user. Otherwise, the function returns `null` and sysError is set to one of these values:

`-1` User canceled the dialog box.
`-2` Insufficient memory.
`-3` Invalid window handle passed to function.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING chooseDirectoryDlg(STRING,STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `chooseDirectoryDlg32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<caption text>`	A string that contains the text displayed in the dialog box caption bar.
`<default path>`	A string that contains the name of the default path. If `null`, or if an invalid path is specified, the current default directory is used.

EXAMPLES

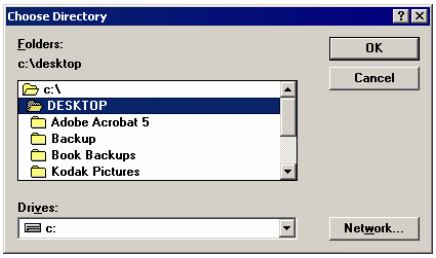
```
to handle buttonClick
    local newDirect
    clear sysError
    newDirect = chooseDirectoryDlg("New Directory",null)
    if newDirect is not null
        get setCurrentDirectory(newDirect)
    else
        if sysError <> -1
            request "Error choosing new directory"
            break to system
        end
    end
end
```

chooseDirectoryDlg32()

TBFILE32.DLL File Function (Attribute)

SYNTAX `chooseDirectoryDlg32(<caption text>,<default path>)`

DESCRIPTION Displays the Choose Directory dialog box (based on the standard Windows Open File dialog box).



RETURNS If no error occurs, `chooseDirectoryDlg32()` returns a string containing the directory chosen by the user. Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    STRING chooseDirectoryDlg32(STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<caption text>`	A string that contains the text displayed in the dialog box caption bar.
`<default path>`	A string that contains the name of the default path. If `null`, or if an invalid path is specified, the current default directory is used.

EXAMPLES

```
to handle buttonClick
    local newDirect
    clear sysError
    newDirect = chooseDirectoryDlg32("New Directory",null)
    if newDirect is not null
        get setCurrentDirectory(newDirect)
    else
        if sysError <> -1
            request "Error choosing new directory"
            break to system
        end
    end
end
```

chooseDirectoryDlgLFN()

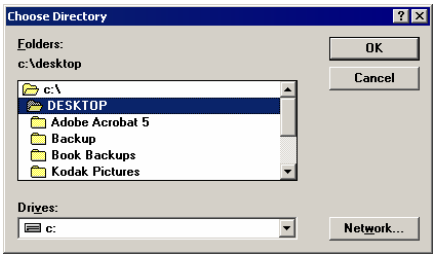
TBDLG.DLL File Function (Dialog)

SYNTAX

```
chooseDirectoryDlgLFN(<caption text>,<default path>)
```

DESCRIPTION

Displays the Choose Directory dialog box (based on the standard Windows Open File dialog box), and returns a long file name version of the directory chosen.



RETURNS

If no error occurs, `chooseDirectoryDlgLFN()` returns a string containing the directory chosen by the user. Otherwise, the function returns `null` and sysError is set to one of these values:

`-1` User canceled the dialog box.
`-2` Insufficient memory.
`-3` Invalid window handle passed to function.

NOTES

You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING chooseDirectoryDlgLFN(STRING,STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `chooseDirectoryDlg32()` instead.

PARAMETERS

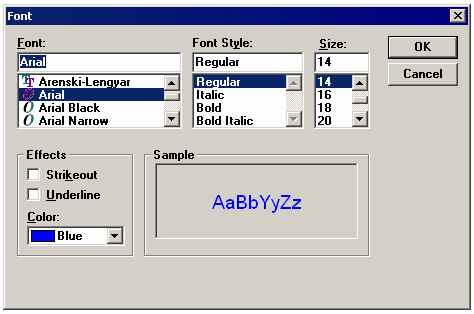
PARAMETER	DESCRIPTION
`<caption text>`	A string that contains the text displayed in the dialog box caption bar.
`<default path>`	A string that contains the name of the default path. If `null`, or if an invalid path is specified, the current default directory is used.

EXAMPLES

```
to handle buttonClick
    local newDirect
    clear sysError
    newDirect = chooseDirectoryDlgLFN("New Directory",null)
    if newDirect is not null
        get setCurrentDirectory(newDirect)
    else
        if sysError <> -1
            request "Error choosing new directory"
            break to system
        end
    end
end
```

SYNTAX `chooseFontDlg(<setup options>,<style options>)`

DESCRIPTION Displays the standard Windows Font dialog box.



RETURNS If no error occurs, `chooseFontDlg()` returns a list of the user's selections. Otherwise, the function returns `null` and sets `sysError` to one of these values:

`-1` User canceled the dialog box.
`-2` Memory allocation error.
`-3` Illegal setup string.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING chooseFontDlg(STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<setup options>`	A string with a comma-separated list of available font options. The syntax of the string of values for `<setup options>` is: `<font face>,<font size>,<red value>,<green value>,<blue value>,<style string>[,strikeout][,underline]` Values for the first six parameters are always required; the last two parameters are optional.

LIST ITEM	DESCRIPTION
`<font face>`	Name of the selected font. If user cancels the dialog box, this item returns `null`.
`<font size>`	Selected `point size`.
`<red value>`	Red component of the default `RGB` color (a value between `0` and `255`).
`<green value>`	Green component of the default `RGB` color (a value between `0` and `255`).
`<blue value>`	Blue component of the default `RGB` color (a value between `0` and `255`)
`<font style>`	String of one or more of the following values: `regular`, `bold`, `italic`, or `bold italic`.
`<optional style1>`	`strikeout`
`<optional style2>`	`underline`

PARAMETER	DESCRIPTION
<style options>	A string with a comma-separated list of available style options. You can use any combination of the values listed in the following table.

LIST ITEM	DESCRIPTION
ansiOnly	List only ANSI fonts.
forceExists	Prompts user to choose a valid font face or font style to exit the dialog box. User must click OK to exit the dialog box.
noGDI	Do not list simulated fonts.
noVector	Do not list vector fonts.
printer	List fonts that can be printed by the selected printer.
scalableOnly	List only scalable fonts.
screen	List fonts that can be displayed onscreen.
trueTypeOnly	List only Apple TrueType fonts.
WYSIWYG	List only fonts that can be both displayed onscreen and printed by the selected printer.

EXAMPLES

```
to handle buttonClick
    fnt = chooseFontDlg("courier,10,255,0,0,regular,underline", \
        "trueTypeOnly")
    if fnt is null
        request "User canceled."
    else
        fontFace of self = item 1 of fnt
        fontSize of self = item 2 of fnt
        rgbStroke of self = items 3 to 5 of fnt
        if item 6 of fnt is "regular"
            clear item 6 of fnt
        end
        fontStyle of self = items 6 to itemCount(fnt) of fnt
    end
end
```

clear

Editing

DESCRIPTION

Sent to the page when `Delete` is chosen from the `Edit` menu.

You can also `send` this message using the `send clear` statement. ToolBook's default response is to delete the current selection *without* placing it on the `Clipboard` or, if there is no selection, to do nothing.

Deleting an object sends a `destroy` message to that object. To prevent an object from being destroyed, write a handler for the `destroy` message that includes a `break to system` statement.

To disable the `clear` message in an application, write a handler for it that includes only `to handle` and `end` statements.

NOTE

As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
--Turns the mouse pointer into an "object killer"
to handle buttonClick
    focus = null
    select target
    send clear
end

--Disables the clear statement, so the object cannot be deleted
to handle clear
end
```

clear

Assigning Values

SYNTAX

```
clear <object> [of <container>]
clear <container>
clear <array name>
```

DESCRIPTION

Deletes an object or the contents of a container.

NOTES

Clearing a textline also clears the `CRLF` at the end of the textline.

You can use the clear command to delete a fixed or dynamic array. After an array is cleared, it is no longer accessible in the current handler. Use the clear command to free the memory used by a large array.

To preserve the dimensions of a dynamic array while clearing its elements, use the `fill` command with a `null` value.

If you attempt to use the `clear` command with a property name that is not a standard property, ToolBook will search the object hierarchy for a corresponding `to set` handler for that property name. If no such handler exists, ToolBook then treats the name as a `user property` and clears its value.

To delete an object you can also use the `deleteObject()` function.

PARAMETERS

PARAMETER	DESCRIPTION
`<object>`	A reference to an object.
`<container>`	A reference to a container. If `<container>` is a reference to a typed variable, ToolBook clears it and sets its value to `null`.
`<array name>`	The name of an array.

EXAMPLES

```
clear x

clear text of field "story"

clear line "connector" of page "connect the dots"
```

clientFromPage()

TBWIN.DLL Screen Display Function

SYNTAX

```
clientFromPage(<pageScroll>,<magnification>,<point | rectangle>)
```

DESCRIPTION

Converts a set of coordinates from ToolBook `page units` to coordinates in `pixels` relative to the upper-left corner of the `client area` of the ToolBook window.

The client area is defined as the working area below the menu bar, not including the scroll bars.

This function is very useful for positioning and sizing a child viewer.

RETURNS If no error occurs, `clientFromPage()` returns the coordinates of either a `point` or a `rectangle` according to the value of `<point|rectangle>`.

Otherwise, the function returns `null` and `sysError` is set to a negative value:

`-20` Memory allocation error.
`-99` First or last parameter was invalid.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    STRING clientFromPage(STRING,INT,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<pageScroll>`	The value of `pageScroll` for a viewer such as the `mainWindow`, this window, or a valid viewer reference.
`<magnification>`	The value of `magnification` for a viewer such as the mainWindow, this window, or a valid viewer reference.
`<point|rectangle>`	For a `point`, a list containing two numbers (x and y coordinates). For a `rectangle`, a list containing four numbers (coordinates of upper-left and lower-right corners of the rectangle).

EXAMPLES

```
-- Position the help viewer over the top of a placeholder field
bounds of viewer "help" = clientFromPage(pageScroll of mainWindow, \
  magnification of mainWindow,bounds of field "placeholder")
```

clientFromScreen()

TBWIN.DLL Screen Display Function

SYNTAX `clientFromScreen(<windowHandle>,<point|rectangle>)`

DESCRIPTION Converts a `point` or `rectangle` in `pixels` relative to the upper-left corner of the `screen` into a `point` or `rectangle` in `pixels` relative to the upper-left corner of the `client area`.

The client area is the working area below the menu bar, not including the scroll bars.

RETURNS If no error occurs, `clientFromScreen()` returns the coordinates of either a `point` or a `rectangle`, depending on the last parameter.

Otherwise, the function returns `null` and `sysError` is set to a negative value:

`-20` Memory allocation error.
`-30` `<windowHandle>` was invalid.
`-99` `<point|rectangle>` was invalid.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    STRING clientFromScreen(WORD,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<windowHandle>	The value of windowHandle of a viewer such as the mainWindow, this window, or a valid viewer reference.
<point|rectangle>	For a point, a list containing two numbers (x and y coordinates). For a rectangle, a list containing four numbers (coordinates of upper-left and lower-right corners of the rectangle).

EXAMPLES

```
to handle buttonClick
    get clientFromScreen(windowHandle of this window,"0,0")
    request "The corner of the screen is at" && IT && \
      "(in pixels) relative" && "to the corner of the client area."
end
```

clientHandle

Property

DESCRIPTION A property of a viewer that specifies the 16-bit window handle of a viewer's client window.

NOTES This property cannot be set.

Various functions in ToolBook require the passing of a parameter that represents the window handle to a ToolBook viewer, or client area.

For example the translateWindowMessage control structure requires a window handle.

VALUES A 16-bit value assigned by Windows when the viewer is opened.

EXAMPLES

```
to handle initTranslation
    translateWindowMessage for clientHandle of viewer "help"
        on 0x0203 send properties to this book
    end
end
```

clientHandle32

Property

DESCRIPTION A property of a viewer that specifies the 32-bit window handle of a viewer's client window.

NOTES This property cannot be set.

Various functions in ToolBook require the passing of a parameter that represents the window handle to a ToolBook viewer, or client area.

VALUES A 32-bit value assigned by Windows when the viewer is opened.

EXAMPLES

```
parentHandle32 of viewer "help" = clientHandle32 of mainWindow
```

clientSize

Property

DESCRIPTION A property of a viewer that specifies the size of a viewer's client window in page units.

NOTES You can get or set this property.

VALUES A list of two positive whole numbers in page units that specify width and height of the clientSize of a viewer.

EXAMPLES

```
clientSize of viewer "help" = 3000,8000
```

clientToPageUnits()

Screen Display Function

SYNTAX `clientToPageUnits(<coordinates>)`

DESCRIPTION Converts `pixels` relative to the client window into `page units`.

Use this function to align an object on a page relative to a window that is a child of a client window (such as a child window that isn't tiled).

This function accounts for the current values of the viewer's `pageScroll` and `magnification` properties.

NOTES You can also use this function to determine how large an object needs to be to fit within a specified number of `pixels` on the current display device.

The conversion of client-relative `pixels` to `page units` is dependent upon the currently installed video driver.

PARAMETERS

PARAMETER	DESCRIPTION
`<coordinates>`	A list of numbers in `pixels` that specify a position or positions on the screen. You can include as many numbers as necessary, but they must be in pairs. You can refer to the `bounds`, `position`, `size`, or `vertices` property as the value for `<coordinates>`.

clientToScreen()

Screen Display Function

SYNTAX `clientToScreen(<coordinates>,<viewer reference>)`

DESCRIPTION Converts `pixels` relative to the specified viewer's client window into `pixels` relative to the origin of the screen.

NOTES Use this function to position a popup window over a window that is a child of the client window (such as a child window that isn't tiled).

PARAMETERS

PARAMETER	DESCRIPTION
`<coordinates>`	A list of numbers in `pixels` that specify a position or positions on the screen. You can include as many numbers as necessary, but they must be in pairs. You can refer to the `bounds`, `position`, `size`, or `vertices` property as the value for `<coordinates>`.
`<viewer reference>`	A valid viewer reference. If the specified viewer is not open, ToolBook displays an error message.

clipboardFormats()

Clipboard Function

SYNTAX `clipboardFormats()`

DESCRIPTION Returns a list of the Clipboard formats that are supported by ToolBook and currently available on the Clipboard.

All available formats are listed even if the current selection or current focus context does not allow the formats to be pasted.

PARAMETERS No Parameters

EXAMPLES

```
if clipboardFormats() contains "text"
    send pasteSpecial "text"
end
```

close

Viewer Command

SYNTAX `close <viewer reference>`

DESCRIPTION Closes an instance of a viewer. If the viewer is `visible`, it is `hidden` before its instance is destroyed.

All viewers close when a user exits an instance of ToolBook.

NOTES The `close` command causes the `closeWindow` message to be sent. This command cannot be used with ToolBook's Main window (`viewer id 0`).

PARAMETERS

PARAMETER	DESCRIPTION
`<viewer reference>`	A valid reference to a viewer object or a list of viewers.

EXAMPLES

```
to handle buttonClick
    close this window
end
```

closeFile

File Manipulation Command

SYNTAX `closeFile <file name>`

DESCRIPTION Closes a file previously opened with the `openFile` or `createFile` command.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	A file name, including the path if necessary.

EXAMPLES

```
closeFile fName
```

```
closeFile "readme.txt"
```

closeWindow

Notification Message

DESCRIPTION Sent to a viewer just before it is closed.

A viewer can close as a result of the `close` command or when the user exits from the current instance.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

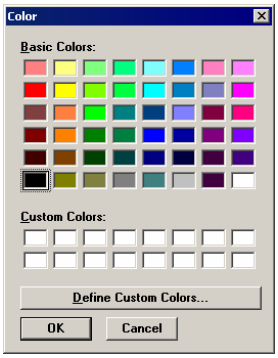
```
to handle closeWindow
    selectedTextlines of field "fieldNames" = null
    enabled of button "OK" = false
    forward
end
```

colorPaletteDlg()

TBDLG.DLL Color Functions

SYNTAX `colorPaletteDlg(<red value>,<green value>,<blue value>)`

DESCRIPTION Displays the standard Windows Color dialog box.



RETURNS If no error occurs, `colorPaletteDlg()` returns the `RGB` value of the color chosen by the user. Otherwise, the function returns `null` and `sysError` is set to one of these values:

`-1` User canceled the dialog box.
`-2` Memory allocation error.

To assign a color returned from the `colorPaletteDlg()` function to an object, use the `rgbStroke` and `rgbFill` properties.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING colorPaletteDlg(BYTE,BYTE,BYTE)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<red value>`	Red component of the default `RGB` color (a value between `0` and `255`)
`<green value>`	Green component of the default `RGB` color (a value between `0` and `255`)
`<blue value>`	Blue component of the default `RGB` color (a value between `0` and `255`)

EXAMPLES

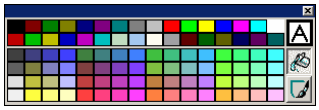
```
get colorPaletteDlg(255,255,255)
rgbFill of rectangle "x2" = IT
```

colorTray

System Object

DESCRIPTION The object type name for the `Color Tray`, which is used to apply color to selected objects.

NOTES To open the Color Tray, use the `show colorTray` statement. You can also show the Color Tray by clicking its button on the tool bar.



The colors displayed in the lower half of the Color Tray are specified in the book's `customColors` property. You can change these colors by double-clicking a custom color in the Color Tray.

Use the `hide`, `show`, and `move` commands to control the `visibility` or `position` of the Color Tray.

The Color Tray cannot be shown in Runtime ToolBook.

PROPERTY	VALUES
`bounds`	List of four whole numbers in `pixels`.
`position`	List of two whole numbers in `pixels`.
`vertices`	List of four whole numbers in `pixels`.
`visible`	`True` or `false`.

SYNTAX `draw comboBox from <location> to <location>`

DESCRIPTION The object type name for `combobox`.

NOTES The parent of a combobox can be a page, background, or group.

TO REFER TO A COMBOBOX	USE THIS SYNTAX
On the current page	`comboBox "Topic"` `comboBox ID 3`
In a group	`comboBox "Styles" of group "Options"`
On another page	`comboBox "musicType" of page 5`
On another background	`comboBox "pageNum" of background "x3"`
In another book	`comboBox "Styles" of page "Fields" \` `  of book "App.tbk"`

You can quickly find object references for a combobox by right-clicking the combobox to display its right-click menu. You can also set certain properties for a combobox through its right-click menu.

The following messages are sent to comboboxes:

MESSAGE	WHEN SENT
`enterComboBox`	Sent when the combobox receives the focus.
`leaveComboBox`	Sent when the focus leaves the combobox.
`enterDropDown`	Sent when user activates drop-down list, just before the list is displayed.
`leaveDropDown`	Sent just after the list is hidden.
`selectChange`	Sent each time the user selects an item in the combobox.

The appearance and state of any particular combobox are controlled by its `dropDownItems`, `enabled`, `editable`, `lineCount`, and `scrollable` properties; by the `focus` property; and by the user's actions.

When the `editable` property of a combobox is `true`, users can type text in the edit field of the combobox; otherwise they cannot.

A combobox's `enabled` property specifies whether the combobox can receive the input focus or mouse event messages at `Reader` level. When a combobox's `enabled` property is set to `false`, the combobox is `disabled`; it cannot receive keyboard input or mouse event messages and is excluded from the tabbing order.

If the combobox's `scrollable` property is `true`, the drop-down list box has a scroll bar. At `Reader` level, scroll bars do not appear unless the number of items in the drop-down list box exceeds the space provided in the list box and the value for `dropDownItems` is greater than the value for `lineCount`.

To add text to the drop-down list box at `Reader` level, set the `dropDownItems` property to a string of text.

To automatically sort items alphanumerically in the drop-down list box, set the `sortItems` property to `true`.

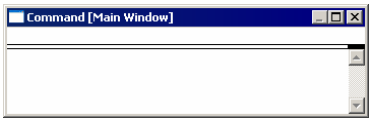
commandWindow

System Object

DESCRIPTION The object type name for the `Command Window`, which is used to execute script statements and display results outside the context of a handler.

NOTES When you type an expression in the Command window and press Enter, ToolBook attempts to execute the statement.

To open the Command window, click the Command window button on the tool bar, or choose Command from the View menu, or execute the `show commandWindow` statement in a script.



You cannot display the Command window in Runtime ToolBook.

The Command window includes a listing of the last `20` statements that were executed from the Command window.

The caption in the Command window's title bar indicates which viewer is active (the focus window) for the Command window.

These commands can directly affect the Command window: `clear`, `hide`, `move`, `put`, `set`, and `show`.

PROPERTY	VALUES
`bounds`	List of four whole numbers in `pixels`.
`position`	List of two whole numbers in `pixels`.
`size`	List of two whole numbers in `pixels`.
`vertices`	List of four whole numbers in `pixels`.
`visible`	`True` or `false`.

compoundFactor()

Financial Values

SYNTAX `compoundFactor(<rate>,<periods>)`

DESCRIPTION Returns the future value of an interest-bearing account.

PARAMETERS

PARAMETER	DESCRIPTION
`<rate>`	Any expression that yields a number representing the rate per period expressed as a decimal fraction.
`<periods>`	Any expression that yields a number representing the number of periods over which the value is calculated and compounded.

EXAMPLES

```
--Calculates compound factor at 10.375% for 12 months
x = compoundFactor(.10375/12, 12)
```

SYNTAX

```
conditions
    when <expression>
        <statements>
    [when <expression>
        <statements>...]
[else
    <statements>]
end [conditions]
```

DESCRIPTION

Executes statements in the body of the control structure based on the value of an expression.

NOTES

The `conditions` control structure includes `1` or more `when` statements and an optional `else` statement.

ToolBook evaluates the `conditions` statement by sequentially evaluating each `when` clause from top to bottom and stopping when it finds an expression that is `true`.

The `statements` associated with the first `when` clause that evaluates to `true` are executed, then control is passed to the statement following the `end [conditions]` statement.

If no `when` expression is `true` and there is an `else` clause, the `statements` associated with the `else` clause are executed.

If there is no `else` clause, control passes to the statement following the `end [conditions]` statement.

If a `when` clause evaluates to `true`, control is passed to the first executable statement in the conditions block.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that evaluates to `true` or `false`.
`<statements>`	One or more OpenScript statements.

EXAMPLES

```
-- Sample #1
conditions
    when val = 1
        request "Correct!"
    when val = 2
        request "Close, but not correct."
        position of paintObject "arrow" = 0,0
        step k from 0 to 4000 by 15
            item 1 of position of paintObject "arrow" = k
        end
end

-- Sample #2
conditions
    when val = 1
    when val = "one"
    when val = "#1"
        request "Correct!"
    when val = 2
    when val = "two"
    when val = "#2"
        request "Close, but not correct."
else
    request "Sorry, you were not even close.
end
```

contains

Logic Operators

SYNTAX `<expression> contains <expression>`

DESCRIPTION Returns `true` if the right expression is found within the left expression. Otherwise returns `false`.

NOTES This is a string comparison operator. If you attempt to use numbers, their string equivalents will be compared.

This operator is not case sensitive.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	Any expression.

EXAMPLES

```
if text of field "password" contains 123456
    go to page "welcome"
end
```

contents

Help

DESCRIPTION Sent to the page when `Contents` is chosen from the `Help` menu, or when `F1` is pressed.

You can also `send` this message using the `send contents` statement. ToolBook's default response is to display the list of contents in the ToolBook `Help` file.

If you would like to have `F1` launch your own help file you will need to handle the `contents` message and in turn launch your own help file using the `ASYM_WinHelp()` function.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle contents
    get ASYM_WinHelp(null,"myHelpFile.hlp")
end
```

continue

Program Flow

SYNTAX `continue`

`continue do | while | step [<variable>]`

DESCRIPTION Proceeds to the next iteration of a `do/until`, `while`, or `step` control structure.

When no control structure is specified, the *innermost* control structure is assumed.

PARAMETERS

PARAMETER	DESCRIPTION
`<variable>`	The name of a variable in the enclosing `step` statement. This variable indicates the `step` control structure on which the continue command is to operate. If `<variable>` is omitted from the `continue` step form, the innermost `step` control structure is assumed. If `<variable>` is supplied, ToolBook executes the next iteration of the innermost step statement with the step variable named by `<variable>`.

EXAMPLES

```
-- Ensure all pages have a name
to handle buttonClick
    step k from 1 to pagecount of this book
        curPage = page k
        if name of curPage <> null
            continue
        else
            name of page k = "Not Yet Named"
        end
    end
ene
```

controlStyle

Property

DESCRIPTION

A book property that specifies whether standard control objects in ToolBook appear in the `style` of Windows 3.1 or of Windows 95 and 98.

NOTES

You can get or set this value.

The `style` of objects displayed under Windows 95 and 98 style differs from those under Windows 3.1 in these ways:

OBJECT	DESCRIPTION
All	Disabled objects are not grayed; instead, text on the object appears inset in white.
Windows (viewers)	The maximize and minimize buttons display different graphics, and a Close button appears next to these buttons.
Pushbuttons	Pushbuttons feature a thinner border.
Checkboxes	The graphic for a checkbox that has been checked is a checkmark.
Scrollbars	The graphics for the scroll up and scroll down arrows are different.

VALUES

`Win3`, `Win95`, or `OSDefault`.

The default is `OSDefault`.

If the value of this property is `OSDefault`, ToolBook automatically matches object style to that of the operating environment.

The `Win3` value will not be supported in future releases of ToolBook.

EXAMPLES

```
--Sets style of controls to Windows 3.1
controlStyle of this book = "Win3"
```

copy

Clipboard Message

SYNTAX

```
copy [<target>]
```

DESCRIPTION

Sent when `Copy` is chosen from the `Edit` menu.

You can also `send` this message using the `send copy` statement. ToolBook's default response is to place a copy of the current selection on the `Clipboard` or, if there is no selection, to do nothing.

To copy an object using OpenScript, select the object, then send the `copy` message. When you `paste` an object using OpenScript, you should first set `focus` to `null`; ToolBook cannot paste an object if the focus is in a field.

To copy a page using OpenScript, first send the `selectPage` message.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
<target>	An optional parameter that specifies the object to be copied.

EXAMPLES

```
-- Copies an object when it is clicked at Reader level
to handle buttonClick
    if object of target is "page" or object of target is "background"
        break buttonClick
    end
    select target
    send copy
    focus = null
end

-- Copies a ToolBook picture to the Clipboard
send copy picture "eye"
```

copy resource

Resource Commands

SYNTAX `copy resource <resource reference> to <book reference>`

DESCRIPTION Copies resources of any type from one book to another.

PARAMETERS

PARAMETER	DESCRIPTION
<resource reference>	A valid reference to a resource.
<book reference>	A valid reference to another book.

EXAMPLES

```
copy resource icon ID 100 of book "reslib.tbk" to this book

copy resource palette "flesh tones" of this book to book "gallery.tbk"
```

copyFile()

TBDOS.DLL File Function (General)

SYNTAX `copyFile(<source file>,<destination>)`

DESCRIPTION Copies the specified source file and optionally renames the destination file.

RETURNS

1	Function was successful.
0	Undetermined error occurred.
-1	File I/O error occurred.
-8	Source file could not be opened.
-9	Destination file could not be opened.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    INT copyFile(STRING,STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `copyFile32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
<source file>	The path and file name of the file to copy.
<destination>	A valid path, or the name for the copy of the source file. If a file name is specified, it may be prefixed with a path.

EXAMPLES

```
-- Copy the lesson file
to handle buttonClick
    fName = "c:\project\information.doc"
    newName = "c:\project\information.bak"
    get copyFile(fName,newName)
end
```

copyFile32()

TBFILE32.DLL File Function (General)

SYNTAX

```
copyFile32(<source file>,<destination>)
```

DESCRIPTION

Copies the specified source file and optionally renames the destination file.

RETURNS

`1` Function was successful.

Otherwise, the function returns one of the values listed in the `TBFILE32 Error Code Table`.

NOTES

As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    INT copyFile32(STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<source file>	The path and file name of the file to copy.
<destination>	A valid path, or the name for the copy of the source file. If a file name is specified, it may be prefixed with a path.

EXAMPLES

```
-- Copy the lesson file
to handle buttonClick
    fName = "c:\project\information.doc"
    newName = "c:\project\information.bak"
    get copyFile32(fName,newName)
end
```

copyObject()

Object Function

SYNTAX

```
copyObject(<object reference>,<destination reference>)
```

DESCRIPTION

Copies a ToolBook object from one location to another without requiring you to navigate to the source location or use the `Clipboard`.

This function cannot copy `hotwords`, `backgrounds`, or `books`.

RETURNS

Returns a reference to the copied object at the specified location.

If an error occurs, `copyObject()` returns `null`.

NOTES If a page is copied, ToolBook inserts it after the page specified as the `<destination reference>`. A page automatically includes its background.

When a recordfield is copied, the text is not included. If a recordfield is copied to the page, it becomes a regular field. If a field that contains a hotword is copied, the hotword is copied as well, but a hotword cannot be copied by itself.

If a viewer is copied, the destination must be a book.

If an object with a resource attached to it, such as a graphic button, is copied to another book, ToolBook automatically copies the resource as well.

PARAMETERS

PARAMETER	DESCRIPTION
`<object reference>`	A reference to a ToolBook object.
`<destination reference>`	A reference to the ToolBook `page`, `background`, `book`, or `group` to which the object should be copied. If a page is copied, a page must be specified in `<destination reference>`. If a viewer is copied, a book must be specified.

EXAMPLES

```
-- Copies a page and changes the name of the new page
get copyObject(this page, page 5)
name of IT = "new page"

-- Copies a viewer to the same book
get copyObject(viewer "forestView", this book)

-- Copies a group to a new page
get copyObject(group "forest" of this page, page "natural resources")

-- Copies a button and places it two inches to the right
-- of the original button
get copyObject(button "chapter1", this page)
item 1 of position of IT = item 1 of position of button "chapter1" + 2440
name of IT = "chapter2"
caption of IT = "Chapter 2"
```

cos()

Trigonometric Values

SYNTAX `cos(<angle>)`

DESCRIPTION Returns the cosine of an angle measured in radians.

The formula for converting degrees to radians is `radians = degrees * (pi/180).`

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<angle>`	An expression that yields a number.

EXAMPLES

```
x = cos(1.047)        -- equals 0.500171
x = cos(60*pi/180)    -- equals 0.5, the cosine of 60 degrees
```

cosh()

Trigonometric Values

SYNTAX `cosh(<number>)`

DESCRIPTION Returns the hyperbolic cosine of an angle measured in radians.

The formula for converting degrees to radians is `radians = degrees * (pi/180)`.

PARAMETERS

PARAMETER	DESCRIPTION
`<number>`	An expression that yields a number.

EXAMPLES

```
x = cosh(1)          -- equals 0.785398 (pi/4 radians)
x = cosh(1)*180/pi   -- equals 45 (degrees)
```

CR

Spacing Constants

DESCRIPTION Represents the carriage return `ANSI` character. This is equivalent to `ANSI 13`. In ToolBook a `CR` by itself is not of much use, but is when used as `CRLF`.

EXAMPLES

```
if text of field "story" contains CR
```

createDirectory()

TBDOS.DLL File Function (Directory)

SYNTAX `createDirectory(<directory name>)`

DESCRIPTION Creates a new directory with the specified name.

RETURNS

`1` Function was successful.
`-3` Function failed.
`-5` Access was denied / sharing violation.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    INT createDirectory(STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `createDirectory32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<directory name>`	The name of the directory to be created.

EXAMPLES

```
get createDirectory("c:\myPersonalProjectFolder")
```

createDirectory32()

TBFILE32.DLL File Function (Directory)

SYNTAX `createDirectory32(<directory name>)`

DESCRIPTION Creates a new directory with the specified name.

RETURNS `1` Function was successful.

Otherwise, the function returns one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    INT createDirectory32(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<directory name>`	The name of the directory to be created.

EXAMPLES

```
get createDirectory32("c:\myPersonalProjectFolder")
```

createFile

File Manipulation Command

SYNTAX `createFile <file name>`

DESCRIPTION Creates a file and leaves the file in an open state as if you performed an `openFile` on it.

NOTES If a file with the specified file name already exists, the contents of that file are deleted.

If a file with the specified file name exists and is a read-only file, `sysError` is set to `read only`.

If you try to create a file when 10 files are already open, or to create a file that is already open, an error occurs and ToolBook displays the `Execution Suspended` message.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	A file name, including the path if necessary.

EXAMPLES

```
-- Copies text of field fField to file fileName
to handle fileField fileName, fField
    createFile fileName
    if sysError is not null
        request "Could not create file" && fileName & "."
        break
    end
    writeFile text of field fField to fileName
    if sysError is not null
        request "Could not write to file" && fileName & "."
    end
    closeFile fileName
end
```

createHotword

Hotword Message

DESCRIPTION Sent to the page when you choose `Create Hotword` from the Text menu.

You can also `send` this message using the `send createHotword` statement. ToolBook's default response is to make the selected text a hotword.

Once the hotword is created the value of `selection` will be the reference to the newly created hotword.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle makeHotword
    if selectedText is not null
        send createHotword
        name of selection = "SEL_1"
    end
end
```

CRLF

Spacing Constants

DESCRIPTION Represents the carriage return `ANSI` character and the linefeed `ANSI` character. This is equivalent to `ANSI 10 ANSI 13`.

When you press Enter while typing into a field, a `CRLF` is inserted into the field. Although you can't see the two individual characters [`CR` and `LF`] you can see their effect, which is to cause the next character to be shown on the next textline.

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES

```
fileName = "c:\pwd.ini"
if fileExist32(fileName) <> 1
    msgText = "Error: The following file is missing:" & CRLF & \
              CRLF & \
              fileName
    request msgText
end
```

currentPage

Property

DESCRIPTION A property of a viewer that specifies the current page displayed in the viewer.

NOTES You can get or set this property.

VALUES A valid page reference.

You cannot set `currentPage` to `null`.

The default is the setting for `defaultPage`. If no value for `defaultPage` exists, `currentPage` is set to the first page of the book that owns the viewer.

EXAMPLES

```
currentPage of viewer "help" = page "OrderFormHelp"
```

cursor

Resource Type

DESCRIPTION The resource type name for a `cursor` resource.

Cursor resources are referenced by `name` or by `ID` with the `cursor` keyword. The `ID` is assigned by the ToolBook system, but you can assign any `name` to the resource.

You can set a `cursor` resource for the `sysCursor` property; for the `normalGraphic`, `invertGraphic`, `disabledGraphic`, and `checkedGraphic` properties of a graphic button; or for the `dragImage` and `noDropImage` object properties.

EXAMPLES

```
normalGraphic of button "exit" = cursor "downArrow"
sysCursor = cursor "wavy"
```

curve

Object Type

SYNTAX

```
draw curve from <location> to <location> to <location>
```

DESCRIPTION The object type name for a `curve`.

NOTES A curve's parent can be a page, background, or group.

The `bounds` and `vertices` properties of a curve do not have the same values. Each set of six values for the `vertices` represents the `x,y` coordinates of the curve's beginning, middle, and end points. The `bounds` are the same as the location of the object's selection handles.

customColors

Property

DESCRIPTION A book property that specifies the `64` colors of the `Color Tray`.

NOTES You can get or set this property.

The `64` colors of the color tray that are controlled by this setting are those below the color tray divider bar. The `2` lines of color above the divider bar cannot be adjusted.

VALUES A string containing `64` textlines, each textline represents one color in the Color Tray as a list of three numbers indicating the `HLS` value of the color.

EXAMPLES

```
-- change the fourth color in the color tray
textline 4 of customColors of this book = 0,20,40
```

cut

Clipboard Message

DESCRIPTION Sent to the page when `Cut` is chosen from the `Edit` menu.

You can also `send` this message using the `send cut` statement. ToolBook's default response is to remove the current selection and place it on the `Clipboard` or, if there is no selection, to do nothing.

To `cut` an object using OpenScript, select the object and `send` the `cut` message. When you `paste` an object using OpenScript, you should first set `focus` to `null` (ToolBook cannot paste an object if the focus is in a field).

To `cut` a page using OpenScript, first send the `selectPage` message.

Cutting an object also sends the `destroy` message. To prevent the object from being destroyed, write a handler for the `destroy` message that includes a `break to system` statement. To disable the cut message, write a handler for it that includes only `to handle` and `end` statements.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
-- Allows user to cut objects by clicking the right mouse button
to handle rightButtonUp
    focus = null
    select target
    send cut
end
```

cyan

Color Constants

DESCRIPTION Represents the color `cyan`. This is equivalent to the `RGB` value of `0,255,255` and the `HLS` value of `180,50,100`.

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES

```
rgbFill of field "list" = cyan
```

```
if rgbStroke of rectangle "drop area" = cyan
    rgbStroke of rectangle "drop area" = 0,191,0
end
```

DATA TYPES

Data Types

DESCRIPTION

TYPE	DESCRIPTION
`int`	16-bit signed integer, range -32768 to 32767
`long`	32-bit signed integer, range -2147483648 to 2147483647
`real`	64-bit floating point number. ToolBook will accept a long value if it is not floating point. All numeric functions and operators that accept real numbers use this type internally
`word`	16-bit unsigned integer, range 0 to 65535
`dword`	32-bit unsigned integer, range 0 to 4294967295
`string`	Character string (Avoid assigning string to data that is more efficiently represented by a different type.)
`logical`	True or false
`point`	x, y; where x and y are integers (a list of two location values)
`color`	List of three non-negative numbers in `HLS` or `RGB` units
`stack`	List of comma-separated items
`date`	Date in system format
`time`	Time in system format
`page`	Explicit reference to a page
`background`	Explicit reference to a background
`layer`	Explicit reference to a page or background
`graphic`	Explicit reference to an object on a page or background
`field`	Explicit reference to a field, record field, or combobox
`object`	Explicit reference to a book, page, background, object, or graphic
`book`	Explicit reference to a book path

ddb()

Financial Values

SYNTAX `ddb(<cost>,<salvage>,<life>,<period>,<factor>)`

DESCRIPTION A financial function that returns the depreciation of an asset for a specific period of time. This function uses the double-declining balance method.

This function is similar to the `syd()` function, but `ddb()` distributes a greater amount of the depreciation in the first year of the asset's life rather than throughout the first few years.

PARAMETERS

PARAMETER	DESCRIPTION
`<cost>`	A number representing the initial cost of an asset.
`<salvage>`	A number representing the value of an asset at the end of its life expectancy (sometimes called the salvage value of an asset).
`<life>`	A number representing the number of periods in the duration of the asset's life expectancy. The number of periods can be in whatever unit you want to use, but must be the same unit used for `<period>`.
`<period>`	A number representing the period for which the depreciation is calculated; <period> must use the same unit as `<life>`.
`<factor>`	A number representing the rate at which the balance declines. The value for `<factor>` is usually `2` or `1.5`. If omitted, the default is `2`.

EXAMPLES

```
-- Calculates the depreciation allowance of a car over ten years
x = ddb(10500,1000,10,10,2)

-- Limits the digits of precision
format number x as "#.##" from sysNumberFormat

-- Puts 281.86 into the field
text of field "Depreciation" = x
```

decrement()

Arithmetic Values

SYNTAX `decrement <expression> [by <amount>]`

DESCRIPTION Subtracts an amount from the value of an expression.

If the value of `<amount>` or value of `<expression>` is not a number ToolBook will generate an `Execution Suspend` error message.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	Any expression that yields a number.
`<amount>`	Any expression that yields a number. Defaults to a value of `1` if not specified.

EXAMPLES

```
to handle buttonClick
    score = 10
    step k from 1 to 10
        if fillColor of rectangle ("rect" & k) <> yellow
            decrement score
        end
    end
    request "Your total score is: " & score
end

-- Move this object to the left by 300 page units
decrement item 1 of position of self by 300
```

defaultAllowDrag

Property

DESCRIPTION A property that specifies an object's default drag-and-drop behavior when the user clicks it at Reader level.

NOTES You can get or set this property.

ToolBook only checks the value of defaultAllowDrag when a to get allowDrag handler is not present in the script of a source object, or further up in the object hierarchy.

ToolBook first calls the to get allowDrag handler, then checks the value of the defaultAllowDrag property.

VALUES True or false; the default is false.

If defaultAllowDrag is false and no to get allowDrag handler is present in the object hierarchy, dragging is not allowed, and ToolBook sends its usual mouse event and button messages, such as buttonClick or buttonDown.

If defaultAllowDrag is true and no to get allowDrag handler is present in the object hierarchy, the beginDrag message is sent and dragging begins.

EXAMPLES

```
-- ensure the objects on this page don't allow dragging
defaultAllowDrag of objects of this page = false
```

defaultAllowDrop

Property

DESCRIPTION A property that specifies an object's default behavior when the cursor enters its bounds during a drag-and-drop operation.

NOTES You can get or set this property.

ToolBook only checks the value of defaultAllowDrop when a to get allowDrop handler is not present in the script of a destination object or further up in the object hierarchy.

ToolBook first calls the to get allowDrop handler, then checks the value of the defaultAllowDrop property.

VALUES True or false; the default is false.

If defaultAllowDrop is false and no to get allowDrop handler is present in the object hierarchy, the object is not a valid drop destination - and the source object's noDropImage or the no-drop system cursor (a circle with a line through it) is displayed.

If defaultAllowDrop is true and no to get allowDrop handler is present in the object hierarchy, the cursor changes to the image specified by the source object's dragImage property when the cursor enters the bounds of the destination object.

ToolBook sends the enterDrop message when the cursor enters the destination object's bounds. The objectDropped message is then sent when the source object is released.

EXAMPLES

```
-- ensure the objects on this page don't allow dropping
defaultAllowDrop of objects of this page = false
```

defaultClientSize

Property

DESCRIPTION A property of a viewer that specifies the default size of the viewer's client window in `page units`.

NOTES You can get or set this property.

The `defaultClientSize` property of a viewer is ignored if the viewer's `autoSize` property is set to `true`.

If the viewer's `autoSize` property is `false` and `defaultClientSize` is `none`, Windows assigns a default size.

VALUES `SizeToPage`, `none`, or a list of two positive integers in `page units` that specify width and height.

The default is `sizeToPage`.

If `defaultClientSize` is set to `sizeToPage`, a viewer's client window is sized to the first page that is displayed.

If `none`, Windows assigns a default size for the client window based on the screen size for popup windows, or the parent window's client area size for child windows.

EXAMPLES

```
defaultClientSize of viewer "help" = 2000,8000
defaultClientSize of viewer "form" = "sizeToPage"
```

defaultPage

Property

DESCRIPTION A property of a viewer that specifies the page that is displayed in the viewer when it is opened.

NOTES You can get or set this property.

Use the `defaultPage` property to specify the page initially displayed in the viewer. Use the `currentPage` property to change the page currently displayed in the viewer while the viewer is already open.

VALUES A page reference to any page in any book.

For the Main window (`viewer id 0`), the page reference must be in the current book.

ToolBook sets the value of the `currentPage` to the value of `defaultPage` when the viewer is opened.

If the value is `null`, the viewer is set to display the first page in its book.

If the value for `defaultPage` is invalid when the viewer is opened, the `currentPage` of the viewer is set to the first page in its book.

EXAMPLES

```
defaultPage of viewer "help" = page "IntroHelpPage"
```

defaultPosition

Property

DESCRIPTION A property of a viewer that specifies the default position of the viewer when it is opened.

NOTES You can get or set this property.

VALUES `Center`, `none`, or a list of two numbers in `pixels` that describe the `x,y` coordinates of the upper-left corner of the viewer's bounding rectangle.

The default is `none`.

If `center` for a popup window, ToolBook centers the viewer on the desktop.

If `center` for a child window, ToolBook centers the viewer within the client area of its parent window.

If `none`, Windows assigns the default position for the viewer and the viewer is cascaded with other viewers.

EXAMPLES

```
defaultPosition of viewer "help" = 0,0
```

defaultState

Property

DESCRIPTION A property of a viewer that specifies the initial state of the viewer when it is opened.

NOTES You can get or set this property.

VALUES `Normal`, `maximized`, `minimized`, or `lockMinimized` can be used for all viewers.

In addition `none` can be used for the main window (`viewer id 0`).

The default is `normal`.

When `defaultState` is set to `lockMinimized`, the viewer always appears in its minimized state.

For example, if you have a book loaded in the Main window, then you load another book in and set `defaultState` of the Main window of the new book equal to `none`, the `state` of the Main window is left as is. If the `defaultState` of `viewer id 0` of the book you load is set to something else, the state of `mainWindow` changes (if necessary) to the new state.

When a viewer's `borderStyle` is `dialogFrame`, ToolBook ignores the settings for `defaultState` and `state`, and only the `Move` and `Close` commands are available from the viewer's Control menu.

The viewer's state is always `normal` if its `borderStyle` property is set to `dialogFrame`.

EXAMPLES

```
defaultState of viewer "help" = "maximized"
```

defaultType

Property

DESCRIPTION A property of a viewer that specifies the default type of the viewer when it is opened.

NOTES You can get or set this property.

VALUES `Popup` or `child`.

The default is `popup`.

A `popup` window floats on top and can move outside of its parent window.

A `popup` window's parent window is initially the Main window (`viewer id 0`), but you can set it to be any window by using the `parentHandle` or `parentWindow` property.

A `child` window is contained within, and clipped by, its parent window.

EXAMPLES

```
defaultType of viewer "help" = "child"
```

delete

Editing

DESCRIPTION Sent to the page when `Delete` is chosen from the `Edit` menu.

You can also `send` this message using the `send clear` statement. ToolBook's default response is to delete the current selection *without* placing it on the `Clipboard` or, if there is no selection, to do nothing.

Deleting an object sends a `destroy` message to that object. To prevent an object from being destroyed, write a handler for the destroy message that includes a `break to system` statement.

To disable the `clear` message in an application, write a handler for it that includes only `to handle` and `end` statements.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
--Turns the mouse pointer into an "object killer"
to handle buttonClick
    focus = null
    select target
    send clear
end

--Disables the clear statement, so the object cannot be deleted
to handle clear
end
```

deleteObject()

Object Function

SYNTAX `deleteObject (<object reference>)`

DESCRIPTION Deletes a ToolBook object.

Afterwards, the object cannot be referenced. If the object is part of a group, the group is resized as appropriate.

RETURNS `True` if the delete was successful, `false` otherwise.

NOTES If a book contains only one page, that one page cannot be deleted, since a book must have at least one page.

If the last page of a background is deleted, the background is also deleted.

A page that is currently being viewed cannot be deleted.

PARAMETERS

PARAMETER	DESCRIPTION
`<object reference>`	A reference to a ToolBook object.

EXAMPLES

```
get deleteObject(button "chapter 1" of page "help")
```

description

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of a book which holds the description entered in the Properties for Book dialog box.

NOTES You can get but not set this property in OpenScript.

You can get but not set this property in the Actions Editor.

In OpenScript, this can also be achieved by getting or setting the `info_Description` property of the book.

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

EXAMPLES

```
x = description of this book
```

destroy

Notification Message

DESCRIPTION Sent to an object just before it is `cleared` or `cut`.

ToolBook sends the `destroy` message to the object that is the `target` of the `clear` or `cut` message. ToolBook sends the `destroy` message to a hotword only when the ToolBook system receives a `removeHotword` message.

ToolBook does not send messages down the object hierarchy, so when a page receives a `destroy` message, the message is not sent to the objects on that page.

To prevent an object from being destroyed, write a handler for the `destroy` message that includes a `break to system` statement.

To disable the `clear` or `cut` message, write a `handler` for either message that includes only to `handle` and `end` statements.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle destroy
    request "Delete the selection?" with "OK" or "Cancel"
    if IT is "Cancel"
        break to system
    else
        forward
    end
end
```

dimensions()

Array Function

SYNTAX

```
dimensions(<array name>)
```

DESCRIPTION Returns a list of dimensions for an array variable. The dimensions of the array variable equal the number of items returned in the list.

NOTES This function is not terribly useful if the array you are checking is a fixed array, since it is likely that you will already know the dimensions for the fixed array.

However for a dynamic array which grows in size as you add items to it, this will be very useful.

PARAMETERS

PARAMETER	DESCRIPTION
<array name>	The name of a declared array variable.

EXAMPLES

```
-- Determine the dimensions of the array 'a' [will return 7,7,7]
local a[][][]
a[1][7][3] = 55
a[7][1][1] = 11
a[2][5][7] = "apple"
request dimensions(a)
```

disable menu

Menu Command/Function

SYNTAX

```
disable menu <name | alias> [in <menu reference> [in <menu reference>]
  ... ] [at <level>]
```

DESCRIPTION

Disables (dims) a menu on the menu bar in the target window. A dimmed menu cannot be chosen by a user.

NOTES

You can disable a submenu by using the in <menu reference> parameter. You can use this parameter multiple times to refer to a cascading menu with multiple submenus.

PARAMETERS

PARAMETER	DESCRIPTION
<name | alias>	The name of the menu, or the alias for the menu.
<menu reference>	A valid reference to an existing menu or submenu in the target window. If more than one menu shares the same reference, ToolBook acts on the first menu from the left on the menu bar.
<level>	Author, Reader, or both; if a level is not specified, the default is the current working level.

EXAMPLES

```
disable menu "Calculations" at both
```

disable menuItem

Menu Command/Function

SYNTAX

```
disable menuItem <name | alias> [in <menu reference> [in <menu reference> ... ]
[at <level>]
```

DESCRIPTION

Disables (dims) a menu item on the menu bar in the target window.

A dimmed menu item cannot be chosen by a user.

NOTES

You can disable a menu item in a submenu by using the in <menu reference> parameter. You can use this parameter multiple times to refer to a cascading menu with multiple submenus.

PARAMETERS

PARAMETER	DESCRIPTION
<name | alias>	The name of the menu item as it appears on the menu, or the alias assigned to the menu item.
<menu reference>	A valid reference to an existing menu or submenu in the target window. If more than one menu shares the same reference, ToolBook acts on the first menu from the left on the menu bar.
<level>	Author, Reader, or both. If a level is not specified, the current working level is assumed.

EXAMPLES

```
disable menuItem "Chapter One" in menu "Go to" at Reader
```

disabledGraphic

Property

DESCRIPTION A button property that specifies a graphic resource assigned to be the resource shown when the button's `enabled` property is set to `false`.

NOTES You can get or set this property.

VALUES A valid resource reference.

You can assign an `icon`, `cursor`, or `bitmap resource`.

If `null`, ToolBook displays the image that is set for the `normal graphic`.

EXAMPLES

```
disabledGraphic of button "Print" = icon "unavailablePrinter"
```

displayAspectX()

TBWIN.DLL Screen Display Function

SYNTAX

```
displayAspectX()
```

DESCRIPTION Returns the relative width of the `pixel` size for the display device.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT displayAspectX()
end
```

PARAMETERS No parameters

EXAMPLES

```
val = displayAspectX()
```

displayAspectY()

TBWIN.DLL Screen Display Function

SYNTAX

```
displayAspectY()
```

DESCRIPTION Returns the relative height of the `pixel` size for the display device.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT displayAspectY()
end
```

PARAMETERS No parameters

EXAMPLES

```
val = displayAspectY()
```

displayAspectXY()

TBWIN.DLL Screen Display Function

SYNTAX `displayAspectXY()`

DESCRIPTION Returns the relative diagonal width of the `pixel` size for the display device.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT displayAspectXY()
end
```

PARAMETERS No parameters

EXAMPLES
```
val = displayAspectXY()
```

displayBitsPerPixel()

TBWIN.DLL Screen Display Function

SYNTAX `displayBitsPerPixel()`

DESCRIPTION Returns the number of color bits for each `pixel` (`color depth`).

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT displayBitsPerPixel()
end
```

PARAMETERS No parameters

EXAMPLES
```
-- My 16 million color display will return a value of 24.
val = displayBitsPerPixel()
```

displayColorPlanes()

TBWIN.DLL Screen Display Function

SYNTAX `displayColorPlanes()`

DESCRIPTION Returns the number of color planes for the device.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT displayColorPlanes()
end
```

PARAMETERS No parameters

EXAMPLES
```
val = displayColorPlanes()
```

displayLogPixelsX()

TBWIN.DLL Screen Display Function

SYNTAX `displayLogPixelsX()`

DESCRIPTION Returns the number of `pixels` per inch for the width of the display device.

Also see `sysPageUnitsPerPixel`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT displayLogPixelsX()
end
```

PARAMETERS No parameters

EXAMPLES

```
-- Normally returns 96 if running with Small Fonts
val = displayLogPixelsX()
```

displayLogPixelsY()

TBWIN.DLL Screen Display Function

SYNTAX `displayLogPixelsX()`

DESCRIPTION Returns the number of `pixels` per inch for the height of the display device.

Also see `sysPageUnitsPerPixel`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT displayLogPixelsY()
end
```

PARAMETERS No parameters

EXAMPLES

```
-- Normally returns 96 if running with Small Fonts
val = displayLogPixelsY()
```

displayFonts()

TBWIN.DLL Font Functions

SYNTAX `displayFonts(<typeface name>)`

DESCRIPTION Queries a list of available sizes for fonts available to be rendered to the screen.

This function will generate font sizes for True Type fonts as well as for Screen fonts.

RETURNS If the parameter is `null`, `displayFonts()` returns a string composed of textlines. Each textline is a list containing the `name` of a typeface and the character sizes available for that typeface.

If the parameter is the name of a typeface, `displayFonts()` returns a list containing the typeface `name` and the character sizes available for that typeface.

If the named typeface is not available, it returns only the name of the typeface.

If an error occurs, the returned value is `null` and `sysError` is set to a negative value.

Example:
`request printerFonts("Arial")` returns this string:
`arial,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48,`
`50,52,54,56,58,60,62,64,66,68,70,72`

Even though `Arial` is a True Type font and is not limited to certain font sizes, the sizes reported back is the string above, ranging between `8` and `72` points.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    STRING displayFonts(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<typeface name>`	The name of the typeface for which you want the available sizes and styles. Use `null` for a list of all typefaces.

EXAMPLES

```
-- Generate a list of all display fonts, but don't show sizes
fnts = displayFonts(null)
step k from 1 to textlineCount(fnts)
    textline k of fnts = item 1 of textline k of fnts
end
text of field "allFontList" = fnts
```

(Divide) div

Arithmetic Operators

SYNTAX `<expression> div <expression>`

DESCRIPTION Mathematically divides the left expression by the right expression.

Only the integer (whole number) portion is returned. Any remainder (partial number) is discarded.

Use the `/` operator if you need to preserve the remainder.

NOTES Dividing by `0` is not a legal mathematical operation. If your right expression results in a 0, you will get an error.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that results in a numeric value.

EXAMPLES

```
ageInYears = totalDays div 365

step k from 1 to (total div 2)

put x div y into z
```

do/until

Control Structure

SYNTAX

```
do
    <statements>
until <expression>
```

DESCRIPTION

Executes the `statements` in the body of the control structure `until` the expression evaluates to `true`.

NOTES

ToolBook checks the value of the `expression` after it executes the statements in the body of the control structure. Consequently, the statements in the body of the control structure are always executed at least once.

Use the `break` command to break out of a `do/until` control structure before it finishes executing.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that evaluates to `true` or `false`.
`<statements>`	One or more OpenScript statements.

EXAMPLES

```
text of field "Instructions" = "Move your mouse over the Apple"
do
    obj = objectFromPoint(sysMousePosition)
until obj = paintObject "apple"
request "Correct"
```

dragImage

Property

DESCRIPTION

A property that specifies the image displayed as the cursor during a `drag-and-drop` operation.

NOTES

You can get or set this property.

The `dragImage` property specifies the cursor that is displayed when the user drags the cursor over an object that accepts a drop.

The `noDropImage` property specifies the cursor that is displayed when it is dragged over an object that does not accept drops.

VALUES

A valid reference to an `icon`, `bitmap`, or `cursor resource`, or `null`; the default is `null`.

If `null`, the default system cursor is used. You can assign either an `icon`, `bitmap`, or `cursor` to this property.

EXAMPLES

```
dragImage of ellipse "b3" = bitmap "star"
```

draw

Object Command

SYNTAX

```
draw <object type> from <location> to <location> [to <location>...]
```

```
draw [with] <object type> [tool] from <location> to <location> [to
  <location>...]
```

DESCRIPTION

Creates an object using the same action as when you select a tool from the tool palette and use the mouse to mark the location of an object's vertices on the page or background.

When you draw an object, ToolBook automatically sets the value of `selection` to the `uniqueName` of the new object.

PARAMETERS

PARAMETER	DESCRIPTION
<object type>	angledLine, arc, button, comboBox, curve, ellipse, field, irregularPolygon, line, ole, pie, polygon, rectangle, recordField, stage, roundedRectangle. If you have added extensions to your book, you can also specify an extension type. The type for each added extension can be determined by passing the mouse pointer over the Tool Palette icon representing the different extensions. The status bar will display the object type as you do so.
<location>	Specifies a point as a list of two numbers in page units: the first represents the distance from the left edge of the page, the second represents the distance from the top of the page. To use the draw command to create a curve, arc, or pie, you must supply three points. To draw an irregular polygon or angled line, you must supply three or more points, up to a maximum of 511; supplying only two points for an irregular polygon or angled line results in a straight line. To use the draw command to create check boxes or radio buttons, which are styles of buttons, you must first use a draw button statement, then set the borderStyle property of the button to checkBox, checkBox3D, radioButton, or radioButton3D.

EXAMPLES

```
draw pie from 1000,100 to 650,874 to 0,0

--Draws and inserts text into label field
--x, y, and fieldName are variables
draw field from x,y to x+1320,y+240
fontFace of selection = Arial
fontSize of selection = 8
textAlignment of selection = right
transparent of selection = true
name of selection = fieldName & "Label"
text of selection = fieldName

--Draws and names a record field
draw recordField from x+1350,y to x+3735,y+240
fontFace of selection = Arial
fontSize of selection = 8
name of selection = rfieldName
```

drawDirect

Property

DESCRIPTION

A object property that specifies the method for drawing the screen image of an object. All object types except hotwords, pages, backgrounds, viewers, and books have this property.

NOTES

You can get or set this property.

Using drawDirect it is possible to have a background image draw/paint itself on top of objects on the foreground.

An object that is drawn directly is drawn onscreen, according to its layer order. Objects that are not drawn directly are drawn offscreen, which means that ToolBook first composes the image in memory and then displays it.

An object that is drawn directly onscreen will always appear to be on top of objects drawn in the offscreen image.

For pages animated with moving objects, set drawDirect to false, otherwise the animation may flicker.

If you create animation by showing and hiding objects or by swapping object layer order, set drawDirect to true for the best effect.

ToolBook can display objects drawn directly more quickly than objects drawn offscreen, but draw direct objects tend to flicker if they are moved.

VALUES `True` or `false`.

The default is the value of `sysDrawDirect` when the object is created.

If `drawDirect` is `true`, the object is drawn directly onscreen.

EXAMPLES
```
drawDirect of button "apple" = true
```

drawTextDirect

Property

DESCRIPTION A field or recordfield property that specifies the method for drawing text in a field.

NOTES You can get or set this property.

Set this property to `true` to improve a field's or recordfield's scrolling speed.

VALUES `True` or `false`; the default is `false`.

If `drawTextDirect` is `true`, the text is drawn directly onscreen even if the field is drawn in the offscreen image.

If `drawTextDirect` is `false`, the effect depends on the value of the field's `drawDirect` property. If `drawDirect` is true, text will be drawn directly; if `false`, the text is drawn offscreen with the field as part of the offscreen image.

EXAMPLES
```
drawTextDirect of field "listing44" = false
```

dropDownItems

Property

DESCRIPTION A combobox property that specifies the contents of the dropdown list box.

NOTES You can get or set this property.

You can have a maximum of `32,000` characters in the dropdown list box.

VALUES A `CRLF` delimited string that represents the text of the items in the dropdown list box.

If no text is in the dropdown list, the value is `null`.

EXAMPLES
```
opts = "TR100" & CRLF & "TR200" & CRLF & "TR300+"
dropDownItems of combobox "partList" = opts
```

editable

Property

DESCRIPTION A combobox property that specifies whether the user can type in the combobox at `Reader` level.

NOTES You can get or set this property.

The appearance of the combobox changes depending on whether its `editable` property is `true` or `false`.

When `editable` is `true`, a space appears between the pushbutton and the field area.

When `editable` is `false`, the pushbutton appears adjacent to the field area.

VALUES `True` or `false`; the default is false.

If `true`, the user can type a custom value into the edit area of the combobox at `Reader` level, or choose an item from the dropdown list box.

If `false`, the user cannot type a value into the field area and must choose an item from the dropdown list box.

eighth

Ordinal Numbers

DESCRIPTION A value used to indicate the eighth item in a sequence or list. Also used to indicate relative position of pages or backgrounds.

EXAMPLES

```
-- The following paired examples show how you can use ordinal
-- references to make OpenScript easier to read. As you can see it is
-- possible to write your scripts with or without ordinal references.
go to the eighth page of this book
go to page 8 of this book

if eighth character of str = "X"
if character 8 of str = "X"

eighth word of text of field "lesson" = "Hello"
word 8 of text of field "lesson" = "Hello"

put "Cancelled:" into eighth character of name of button id 16
put "Cancelled:" into character 8 of name of button id 16
```

ellipse

Object Type

SYNTAX `draw ellipse from <location> to <location>`

DESCRIPTION The object type name for an `ellipse`.

NOTES An ellipse's parent can be a page, background, or group.

You can quickly find object references for an ellipse by right-clicking the ellipse to display its right-click menu. You can also set certain properties for a ellipse through its right-click menu.

For an ellipse, the `vertices` and `bounds` properties have the same value.

enable menu

Menu Command/Function

SYNTAX

```
enable menu <name | alias> [in <menu reference> [in <menu reference>]
  ... ] [at <level>]
```

DESCRIPTION `Enables` a menu if it was previously `disabled` (appears `dimmed`) on a menu bar in the target window.

NOTES You can `enable` a `menu` in a `submenu` by using the in `<menu reference>` parameter. You can use this parameter multiple times to refer to a cascading menu with multiple submenus.

PARAMETERS

PARAMETER	DESCRIPTION
`<name | alias>`	The name of the `menu`, or the `alias` for the menu.
`<menu reference>`	A valid reference to an existing `menu` or `submenu` in the target window. If more than one menu shares the same reference, ToolBook acts on the first menu from the left on the menu bar.
`<level>`	`Author`, `Reader`, or `both`; if a level is not specified, the default is the current working level.

EXAMPLES

```
enable menu "Calculations" at both
```

enable menuItem

Menu Command/Function

SYNTAX

```
enable menuItem <name | alias> [in <menu reference> [in <menu reference>
  ... ] [at <level>]
```

DESCRIPTION

Enables a menu item if it was previously `disabled` on the menu in the target window.

NOTES

You can `enable` a menu item in a submenu using the in `<menu reference>` parameter. You can use this parameter multiple times to refer to a cascading menu with multiple submenus.

PARAMETERS

PARAMETER	DESCRIPTION
`<name | alias>`	The name of the menu item as it appears on the menu, or the alias assigned to the menu item.
`<menu reference>`	A valid reference to an existing menu or submenu in the target window. If more than one menu shares the same reference, ToolBook acts on the first menu from the left on the menu bar.
`<level>`	`Author`, `Reader`, or `both`. If a level is not specified, the current working level is assumed.

EXAMPLES

```
enable menuItem "Chapter One" in menu "Go to" at Reader
```

enabled

Property

DESCRIPTION

A button, combobox, field, or recordfield property, or viewer property that specifies whether the object can receive the `focus` or mouse event messages at `Reader` level.

NOTES

You can get or set this property.

VALUES

`True` or `false`; the default is `true`.

If `true`, the object can receive the `focus` and mouse event messages.

If `false`, the object is `disabled`; it does not receive keyboard input (the `focus`) or mouse event messages such as `buttonClick`, `buttonDown`, or `buttonUp`, and the object is excluded from the tabbing order.

When a button is `disabled` (`enabled` is `false`), its `caption` appears dimmed.

When a combobox is `disabled`, its `pushbutton` appears dimmed.

When a viewer is `disabled`, it cannot become the `active window` or the `focus window`; executing the `activate` command has no effect.

ACTIONS EDITOR

The Actions Editor also supports this feature for the Button object.

EXAMPLES

```
-- prevent print button from working if page is "passwordReset"
notifyAfter enterPage
    enabled of self = (name of this page is not "passwordReset")
end
```

enterApplication

Event Message

DESCRIPTION Sent when a book opens in the ToolBook Main window.

NOTES This message is sent just before the `enterBook` message and just after the `enterSystem` message, to the `page` displayed in the Main window.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle enterApplication
    sysCursor = 4
    show viewer "SplashScreen"
    send initMenus
    send linkMyDLLs
    close viewer "SplashScreen"
    sysCursor = default
    forward -- necessary so that things like logging will work
end
```

enterBackground

Event Message

DESCRIPTION Sent to the `page` when a user first goes to a page with a different background.

NOTES Several actions cause ToolBook to send the `enterBackground` message, including opening a `book`, showing a `viewer`, navigating to a `page` that has a background different from the current one, or executing the `flip`, `go`, or `search` command.

Although you might assume that the `enterBackground` message would be sent to the `background` layer, it is in fact sent to the `page` level.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle enterBackground
    fillColor of this background = sysFillColor
    send setupPage to this page
    forward
end
```

enterBook

Event Message

DESCRIPTION Sent to the current `page` immediately after a book opens in a viewer.

NOTES The `enterBook` message is sent to the current `page`, just after the `enterApplication` message if the viewer is the Main window. Several actions cause the `enterBook` message to be sent, including opening a book for the first time, navigating to another book, or displaying a page or background from another book in a viewer.

If a handler exists for the `enterBook` message, ToolBook displays the Main window, executes the handler, then displays the first page.

If the book is in a new instance, ToolBook executes the handler, displays the Main window, then displays the first page.

If you use an `enterBook` handler to modify the Main window (for example, to adjust its position and size, change its menu bar, or set the `sysLevel` property), set the `sysLockScreen` property to `true` within the handler. This step suppresses screen updates until after the handler finishes executing.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
-- Performs initialization tasks
to handle enterBook
    send initMySystemVars
    if "myLib.sbk" is not in sysBooks
        push "myLib.sbk" onto sysBooks
    end
    forward
end
```

enterButton

Event Message

DESCRIPTION Sent to a button when it receives the `focus`.

NOTES To give the `button` the `focus`, you can tab to the button, click the button, or set the value of focus to the unique name of the button in a script.

The `focus` outline appears in the button unless the button's `excludeTab` property is set to `true`, or its `enabled` property is set to `false`, in which case the button cannot receive the `focus`.

Buttons that are excluded from the tabbing order or buttons that are disabled cannot receive the focus, so they never receive the `enterButton` message.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle enterButton
    fillColor of self = gray
    forward
end

to handle leaveButton
    fillColor of self = lightGray
    forward
end
```

enterComboBox

Event Message

DESCRIPTION Sent to a combobox when it receives the `focus`.

NOTES To give the `combobox` the `focus`, you can tab to the button, click the combobox, or set the value of focus to the unique name of the combobox in a script.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
-- Sets up combobox items when combobox gets the focus
to handle enterComboBox
    my dropDownItems = getFileList("*.tbk")
    forward
end
```

enterDropDown

Event Message

DESCRIPTION Sent to a `combobox` when the user clicks the combobox `pushbutton`.

NOTES The `enterDropDown` message is sent just before the drop-down list box appears.

Use this message to set the value for `dropDownItems` viewed in the drop-down list box.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
-- Sets up combobox items
to handle enterDropDown
    my dropDownItems = getFileList("*.tbk")
    forward
end
```

enterField

Event Message

DESCRIPTION Sent to a field when it receives the focus.

NOTES To give the `field` the `focus`, you can tab to the field, click the field, or set the value of focus to the unique name of the field in a script.

ToolBook does not send the `enterField` message if a field's `activated` property is `true` because the `focus` cannot be set to the field.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
-- Puts the field's text into a Help field
to handle enterField
    --pHelpText is a user property that contains Help text
    hlp = pHelpText of target
    if hlp <> null
        text of field "FieldHelp" = hlp
    end
    forward
end
```

enterMenu

Event Message

SYNTAX `enterMenu <menu name>,<menu alias>`

DESCRIPTION Sent whenever a menu is selected.

NOTES The `enterMenu` message is sent to the `viewer` that owns the menu bar with the selected menu.

You can use the `enterMenu` message to control the appearance of a menu before the menu is actually displayed.

Mouse events or keyboard input can also cause the `enterMenu` event message to be sent.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
<menu name>	The name of the menu that has been selected.
<menu alias>	The alias of the menu that has been selected.

EXAMPLES

```
-- Checks current selection for group to enable menu item "editGroup"
to handle enterMenu pMenu, pAlias
    conditions
        when pAlias = "editGroup"
            if selection <> null and object of selection = "group"
                enable menuItem "EditGroup"
            else
                disable menuItem "EditGroup"
            end
        when pAlias = "editUserProperties"
            -- other menu options here
    end
    forward
end
```

enterPage

Event Message

DESCRIPTION Sent to the page when the user navigates to a page or opens the book to a specific page, or when another action causes ToolBook to navigate to a page.

EXTRA NOTE As with most all system generated messages, it is important to forward the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle enterPage
    caption of this window = name of this page
    forward
end
```

enterRecordField

Event Message

DESCRIPTION Sent to a recordfield when it receives the focus.

NOTE To give the recordfield the focus, you can tab to the field, click the field, or set the value of focus to the unique name of the field in a script.

ToolBook does not send the enterRecordField message if a field's activated property is true because the focus cannot be set to the field.

EXTRA NOTE As with most all system generated messages, it is important to forward the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle enterRecordField
    if name of target is "userName"
        move group "fieldPointer" to 3000,1250
        fillColor of group "fieldPointer" = blue
    end
    forward
end
```

enterSystem

Event Message

SYNTAX `enterSystem <cmdLine>`

DESCRIPTION Sent to the current page of the Main window when a new instance of ToolBook opens.

NOTE The `enterSystem` message is sent before the `enterApplication` message.

The `<cmdLine>` parameter contains the book's command line exactly as you type it as an argument to the `enterSystem` message. For example, if the instance is started with the command line `INSTRUCTOR.EXE LOGIN.TBK JOHN`, the `<cmdLine>` parameter would contain `LOGIN.TBK JOHN`.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
`<cmdLine>`	A reference to the command line that is used to start the ToolBook instance.

EXAMPLES

```
-- If a file name is specified in the command line,
-- opens the file automatically
to handle enterSystem cmdLine
    numWord = wordCount(cmdLine)
    conditions
        when numWord > 1
            send openMyFile word 2 of cmdLine
        when numWord = 0
            caption of mainWindow = space
    end
    forward
end
```

enterWindow

Event Message

SYNTAX `enterWindow <old viewer>,<old handle>`

DESCRIPTION Sent to a `viewer` when it is activated.

NOTE The viewer becomes the active window when a user clicks it, when the `activate` command is executed, or when the `focusWindow` property is set to a visible viewer.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
`<old viewer>`	A valid reference to the `viewer` that last held the focus. `Null`, if the window that last held the focus is not a ToolBook viewer.
`<old handle>`	The `window handle` of the viewer or non-ToolBook window that last held the focus.

EXAMPLES

```
to handle enterWindow
    focus = field "FirstName"
    forward
end
```

(End Of File) EOF

Special Constants

DESCRIPTION Represents the end of a file. Used in combination with `readFile`.

EXAMPLES

```
-- Reads to the end-of-file character
readFile "myFile.txt" to EOF
text of field "allText" = IT
```

evaluate()

Script Control

SYNTAX `evaluate(<expression>)`

DESCRIPTION Evaluates an expression and returns the result.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	Any expression.

EXAMPLES

```
dataList = "fName,lName,age"
step k from 1 to itemCount(dataList)
    curItem = item k of dataList
    get evaluate(curItem && "of button" && quote & "Data" & quote)
    request IT
end
```

excludeTab

Property

DESCRIPTION A button property that specifies whether a button can receive the focus as a result of keyboard actions, mouse clicks, or script statements at `Reader` level.

NOTES You can get or set this property.

When `excludeTab` is `true`, the button cannot get the focus when the user presses Tab, and therefore the button's script cannot run as a result of keyboard actions. However, you can still set focus to the unique name of a button through script statements and mouse clicks.

If the caption property of a button without a graphic is `null` and `excludeTab` is `false`, the focus outline does not appear in the button. However, a user can press Tab to move the focus to that button.

If the caption property of button with a graphic is `null`, the focus outline appears around the graphic.

VALUES `True` or `false`.

The default is `false`.

EXAMPLES

```
excludeTab of button "RJ11" = false
```

execute

Script Control

SYNTAX `execute <source>`

DESCRIPTION Executes `<source>` as one or more OpenScript statements in the context of the current handler.

The `execute <source>` command runs more slowly than executing the same statements in a script but is useful in cases where you can't otherwise specify a statement.

NOTES You can run multi-line scripts in an execute statement by using a `CRLF` or semicolon (`;`) as a delimiter.

PARAMETERS

PARAMETER	DESCRIPTION
<source>	A reference to one or more OpenScript statements. The <source> parameter can include the name of a container that holds OpenScript statements, or a string combining one or more OpenScript statements.

EXAMPLES

```
execute text of field "MyScript"

-- Uses execute to set a user property whose name is in a variable
propName = "car"
value = "123"
cmd = propName && "of this book =" && value
execute cmd
```

exit

Program Termination

DESCRIPTION

Sent to the page when `Exit` is chosen from the `File` menu or `Close` is chosen from the `Control` menu.

You can also `send` this message using the `send exit` statement. ToolBook's default response is to end the ToolBook session.

ToolBook does not respond to the `exit` message until it finishes executing the `topmost` calling handler. To exit ToolBook immediately from a script, first include the statement set `sysSuspendMessages` to `true`, then `send exit`, followed by the statement `break to system`.

NOTE

As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle buttonClick
    if processDone() <> true
        request "Do you really want to exit?" with "Yes" or "No"
        if IT = "yes"
            sysSuspendMessages = true
            send exit
            break to system
        end
    end
end
```

exp()

Logarithmic Values

SYNTAX

`exp(<number>)`

DESCRIPTION

Returns `e` raised to the power of `<number>`.

The constant `e` equals `2.71828182845904`, the base of the natural logarithm.

ACTIONS EDITOR

The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<number>	An expression that yields a number.

EXAMPLES

```
x = exp(1)      -- equals 2.718282 (the approximate value of e)
x = exp(2)      -- equals e2, or 7.389056
x = exp(ln(3))  -- equals 3
```

expandString()

SYNTAX `expandString(<string image>,<param>[,<param>])`

DESCRIPTION Replaces placeholders in a string with parameters.

RETURNS Returns the modified string.

NOTES This function works identically to the function: `ASYM_ExpandString()`

PARAMETERS

PARAMETER	DESCRIPTION
`<string image>`	A string that can contain placeholders to be replaced by parameters. Placeholders are indicated in the form `%<n>`, where `<n>` corresponds to the parameter number. For example `%1` represents the first parameter, `%2` represents the second parameter, and so on. If by chance you want to include the `%` character in your original string, and not have it represent a replacement parameter, use two `%` characters instead of one. Example: `x = expandString("Buy 2%% milk at the %1","market")` The same placeholder can appear more than once in the `<string image>`. If the parameter referenced by a numbered placeholder does not exist, the placeholder is replaced by a `null` string.
`<param>`	A string or expression that evaluates to a string. You can pass up to `99` parameters to this function.

EXAMPLES

```
-- Displays the message: David, you earned 11 points. Great job David!
pts = 11
uName = "David"
msg = "%1, you earned %2 points. Great job %1!"
request expandString(msg,uName,msg)
```

export resource

SYNTAX `export resource <resource reference> as <file name>`

DESCRIPTION Exports a resource to its native file format.

PARAMETERS

PARAMETER	DESCRIPTION
`<resource reference>`	A valid reference to a resource. ToolBook also supports `clip`.
`<file name>`	A string specifying a valid filename with format (`.BMP`, `.GIF`, `.JPG`, `.CUR`, `.ICO`, `.MNU`, `.PAL`, `.TTF`, or `.TXT`) and path for the resource. ToolBook also supports `.CPF` for clip libraries.

EXAMPLES

```
export resource bitmap "clock" as "c:\temp\clock.bmp"

export resource sharedScript "timer" as "c:\temp\script.txt"
```

extend select

Selection Commands

SYNTAX `extend select <objects>`

DESCRIPTION Extends the current selection to include specified objects.

PARAMETERS

PARAMETER	DESCRIPTION
`<objects>`	A list of objects.

EXAMPLES

```
extend select irregularPolygon "doubleLoop"
```

failedAuthorPassword

Property

DESCRIPTION A property of a viewer that specifies whether the user entered the correct password when attempting to go from `Reader` level to `Author` level.

NOTES You can get but cannot set this property.

VALUES `True` or `false`.

If a password is required to enter `author` level and the user types an incorrect password, `failedAuthorPassword` is set to `true`. Otherwise, the value of `failedAuthorPassword` is `false`.

EXAMPLES

```
if failedAuthorPassword = true
    request "Access to Author Mode is denied."
end
```

field

Object Type

DESCRIPTION The object type name for a `field`.

NOTES A field is an object that holds text that can be edited and formatted, and can also be used to create list boxes.

Fields can contain `hotwords`. A field's parent can be a page, background, or group. A field is the parent of any `hotwords` in its text.

TO REFER TO A FIELD	USE THIS SYNTAX
On the current page	`field "Habitat"`
In a group	`field "Names" of group "3D Icon"`
On another page	`field "Help" of page 7`
On another background	`field "Phone" of background "DataForm"`
In another book	`field "Names" of page 10 of book "Records"`
Text of a field	`text of field "Item" of page "Index"`

Fields can be placed on either the `page` or the `background`. A recordfield can be placed only on the background, but its text is stored with the individual page where it is entered in the recordfield.

A field's `enabled` property specifies whether the field can receive the input focus or mouse event messages at `Reader` level. When a field's `enabled` property is set to `false`, the field is `disabled`; it cannot receive keyboard input or mouse event messages and is excluded from the tabbing order.

The `enterField` and `leaveField` messages are sent at `Reader` level when the field gets or loses the focus.

The `textScrolled` notification message is sent to the field when a user clicks the field's scroll bar.

fieldType

Property

DESCRIPTION A field or recordfield property that specifies basic word wrap style and selection behaviors.

NOTES You can get or set this property.

VALUES One of the following: `noWrap`, `singleLineWrap`, `wordWrap`, `multiSelect`, or `singleSelect`.

The default is `wordWrap`.

EXAMPLES

```
fieldType of field "listing" = "multiSelect"
```

fifth

Ordinal Numbers

DESCRIPTION A value used to indicate the fifth item in a sequence or list. Also used to indicate relative position of pages or backgrounds.

EXAMPLES

```
-- The following paired examples show how you can use ordinal
-- references to make OpenScript easier to read. As you can see it is
-- possible to write your scripts with or without ordinal references.
go to the fifth page of this book
go to page 5 of this book

if fifth character of str = "X"
if character 5 of str = "X"

fifth word of text of field "lesson" = "Hello"
word 5 of text of field "lesson" = "Hello"

put "Cancelled:" into fifth character of name of button id 16
put "Cancelled:" into character 5 of name of button id 16
```

fileExists()

TBDOS.DLL File Function (General)

SYNTAX

```
fileExists(<file name>)
```

DESCRIPTION Determines if the specified file exists.

RETURNS

`1`	File exists.
`0`	Operation failed.
`-2`	Specified file was not found.
`-3`	Specified path was invalid.
`-15`	Specified drive was invalid.
`-18`	File name contains wildcards.
`-20`	Memory allocation error.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    INT fileExists(STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `fileExists32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
<file name>	The name of the file to check for. If no path is specified for the file, the current directory is searched.

EXAMPLES

```
to handle buttonClick
    fname = "c:\project\information.doc"
    if fileExists(fName) = 1
        run fName
    else
        request "Sorry, can't location file."
    end
end
```

fileExists32()

TBFILE32.DLL File Function (General)

SYNTAX

```
fileExists32(<file name>)
```

DESCRIPTION

Determines if the specified file exists.

RETURNS

`1` File exists.

Otherwise, the function returns one of the values listed in the `TBFILE32 Error Code Table`.

NOTES

As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    INT fileExists32(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<file name>	The name of the file to check for. If no path is specified for the file, the current directory is searched.

EXAMPLES

```
to handle buttonClick
    fname = "c:\project\information.doc"
    if fileExists32(fName) = 1
        run fName
    else
        request "Sorry, can't location file."
    end
end
```

fileToPrinter()

TBWIN.DLL Printing Function

SYNTAX

```
fileToPrinter(<file name>,<options>,<window handle>,<bShowDlg>)
```

DESCRIPTION

Prints the contents of a raw text file to the current default printer.

This function automatically wraps the text to fit on the printed page, with fixed (8-space) tabs. Only one style of formatting is available for each document.

RETURNS

If successful, the function returns `true`.

Otherwise, it returns `false`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT fileToPrinter(STRING,STRING,WORD,INT)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<file name>	The name of a file.
<options>	A list of textlines (CRLF-separated) that specify the following options: Document Name — A name for the document to be printed. Font Face — A valid font name. Font Style list — Bold, italic, underline, or strikeout. Font Size — A valid font size. Margins in inches — A string that contains a list of four numbers that correspond to the <left>, <top>, <right>, and <bottom> margins.
<window handle>	The window handle of the parent window of the Cancel dialog box.
<bShowDlg>	1 Shows the Cancel dialog box, 0 Does not show the Cancel dialog box.

EXAMPLES

```
to handle buttonClick
    fileN = "c:\master.log"
    docName = "Test"
    fontF = "Times New Roman"
    fontS = "Bold"
    fontSz = 10
    marg = "1.5,1,1.5.1"
    opts = docName & CRLF & fontF & CRLF & fontS & \
           CRLF & fontSz & CRLF & marg
    hndl = windowHandle of mainWindow
    showDlg = 1
    get fileToPrinter(fileN,opts,hndl,showDlg)
end
```

fill

Variable Command

SYNTAX

```
fill <array variable> with <input expression> [in [<parse order>
  [<parse order>] ... ] order]
```

DESCRIPTION Assigns values to an array's elements in a single statement. You can use `fill` to assign a single value to an entire array, or to assign values to one element at a time.

NOTES Use the optional in `[<parse order>] ... ]` order clause to assign individual units of text, such as lines, items, words, or characters, to elements in the array.

Using the `fill` command to initialize an array is much faster and uses less memory than using the traditional looping method, such as a `step loop`.

To preserve the dimensions of a dynamic array while clearing its elements, use the `fill` command with a `null` value, for example: `fill vArray with null`

PARAMETERS

PARAMETER	DESCRIPTION
<array variable>	The name of the array variable.
<input expression>	An expression that specifies the input data. If you have assigned a data type to the array, be sure the data type of the <input expression> matches that of the array. If the <input expression> contains fewer elements than the array, the remaining array items are left un-initialized. If the <input expression> contains more elements than the size of the array, an error results.
<parse order>	A text operator or operators specifying how, and the order in which, the <input expression> is evaluated. The [<parse order>] should contain a text (or "chunk") operator for each dimension of the array. You must enclose each value for this parameter in brackets. Valid text operators are: [textline], [item], [word], or [character]. You can use up to four text operators to subdivide text. The number of text operators must match the dimensions of the array, and the array must be large enough to accept all text components. Otherwise an error results. Each operator you specify breaks the <input expression> text into smaller components and assigns each component to the next dimension in the array. The operators must be used in their logical hierarchical order; that is, textline is the highest level, then item, then word, then character. If one or more [<parse order>] parameters are not included, each element in the array is initialized to the value of the input expression, which must match the data type of the array elements.

EXAMPLES

```
-- Assigns zero to each element in a one-dimensional array
local int vBarrel[]
fill vBarrel with 0

-- Fills a two-dimensional array. Each element in the first dimension
-- of the array is a textline; each element in the second dimension
-- of the array is a word
sampleText = "To be or not to be" & CRLF & "that is the question."
local vInput[2][6]
fill vInput with sampleText in [textline][word] order
```

fillColor

Property

DESCRIPTION

A property of a background, draw object, recordfield, field, stage, button, combobox or graphic object that specifies the object's fill color in HLS values.

NOTES

You can get or set this property.

This property can be used interchangeably with rgbFill, although rgbFill requires RGB color values rather than HLS color values.

When the value of fillColor changes, the value of rgbFill changes as well.

VALUES

A list of three non-negative numbers representing hue, lightness, and saturation, or a valid color constant.

The default is the value of sysFillColor when the object was created.

The valid range for hue is 0 to 360, for lightness, 0 to 100, and for saturation, 0 to 100.

EXAMPLES

```
fillColor of button "Apple" = red

fillColor of button "Apple" = 20,30,40

fillColor of button "Apple" = fillColor of button "Orange"
```

first

Ordinal Numbers

DESCRIPTION A value used to indicate the first item in a sequence or list. Also used to indicate relative position of pages or backgrounds.

EXAMPLES

```
-- The following paired examples show how you can use ordinal
-- references to make OpenScript easier to read. As you can see it is
-- possible to write your scripts with or without ordinal references.
go to the first page of this book
go to page 1 of this book

if first character of str = "X"
if character 1 of str = "X"

first word of text of field "lesson" = "Hello"
word 1 of text of field "lesson" = "Hello"

put "Cancelled:" into first character of name of button id 16
put "Cancelled:" into character 1 of name of button id 16
```

firstIdle

Notification Message

DESCRIPTION Sent when a book first reaches an `idle` state immediately after the page is entered.

Use this message to trigger actions you want to occur after the `enterPage` message is processed by the system.

You can use the `firstIdle` message to set `focus` to an object when entering the page.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
-- Put this in page Script (or higher) to create a timer on the page
to handle firstIdle
    get text of field "timer"
    if IT <> sysTime
        set text of field "timer" to sysTime
    end
    forward
end

-- Place this notify handler in a field to allow the field to
-- get the message and manage its own script
notifyAfter firstIdle
    get my text
    if IT <> sysTime
        set my text to sysTime
    end
    forward
end
```

flip

Navigation Command

SYNTAX

```
flip
flip <number> [pages]
flip all [pages]
```

DESCRIPTION Displays a specified number of pages in the current book in turn, beginning with the current page.

If the specified number of pages is greater than the number of remaining pages, flipping continues from the beginning of the book. The `flip all` form flips all pages, starting and ending at the current page.

NOTE The speed at which the pages flip is not controllable. The pages fill flip as fast as possible for the current machine.

The typical use of the flip command is to preload pages into memory so that later navigation is smoother.

PARAMETERS

PARAMETER	DESCRIPTION
<number>	A positive integer, representing the number of pages to flip.

EXAMPLES

```
-- Place this pair of handlers in the book script to preload pages
-- so they display faster during animation
to handle enterBook
    system FirstTime
    FirstTime = true
    forward
end

to handle enterBackground
    system FirstTime
    if FirstTime is true
        sysLockScreen = true
        flip all
        sysLockScreen = false
        FirstTime = false
    end
    forward
end
```

flipHorizontal

Orientation Message

SYNTAX

```
flipHorizontal [<target>]
```

DESCRIPTION Sent to the page when you choose `Horizontally` from the `Flip` submenu on the `Draw` menu.

You can also `send` this message using the `send flipHorizontal` statement. ToolBook's default response is to horizontally flip the selected object or objects or, if there is no selection, to do nothing.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
<target>	An optional parameter that specifies the object or objects to be manipulated.

EXAMPLES

```
to handle buttonClick
    send flipHorizontal irregularPolygon "bolt"
end

to handle flipHorizontal
    request "You must first be in Edit mode to flip bolts."
end
```

flipVertical

Orientation Message

SYNTAX

```
flipVertical [<target>]
```

DESCRIPTION Sent to the page when you choose `Vertically` from the `Flip` submenu on the `Draw` menu.

You can also `send` this message using the `send flipVertical` statement. ToolBook's default response is to horizontally flip the selected object or objects or, if there is no selection, to do nothing.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
`<target>`	An optional parameter that specifies the object or objects to be manipulated.

EXAMPLES

```
to handle buttonClick
    send flipVertical irregularPolygon "bolt"
end

to handle flipVertical
    request "You must first be in Edit mode to flip bolts."
end
```

floor()

Arithmetic Values

SYNTAX `floor(<number>)`

DESCRIPTION Rounds a number down to the nearest whole number.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<number>`	An expression that results in a number.

EXAMPLES

```
val = 4.56
x = floor(val) -- results in x being set to 4

val = -4.56
x = floor(val) -- results in x being set to -5
```

focus

Property

DESCRIPTION A property of a viewer that specifies which button, field, recordfield, or combobox on the current page or background displayed in a viewer has the `focus`.

If a viewer is not specified, ToolBook uses the viewer referenced by `focusWindow`.

NOTES You can get or set this property.

If a field's `activated` property is true, or if a button's, combobox's, field's, or recordfield's `enabled` property is `false`, that object cannot get the `focus` based on the user's action.

Before executing paste or search commands for any object except text, set the `focus` property to `null` to ensure that the focus is not in a field and that the focus does not control where the script action occurs.

VALUES The `uniqueName` of a button, combobox, field, or recordfield on the current page or background that will receive keyboard event messages at `Reader` level, or `null` if there is no `focus`.

EXAMPLES

```
focus = field "comment"

focus of viewer "form" = combobox "state"
```

focusWindow

System Property

DESCRIPTION A system property that specifies which viewer currently has the focus.

NOTES You can get or set this property.

In most cases, the focus window is the active window.

If a viewer owns one or more child windows, and one of those child windows has the input focus, the focus window is that child window and the active window is its parent.

The value for `focusWindow` and the value for `targetWindow` are usually the same, except in instances when the value of `focusWindow` changes to a different viewer while a script is still running in the target window. In that case, the focus window becomes the target window after ToolBook finishes executing the other viewer's script.

ToolBook displays the name of the viewer that is the focus window within brackets in the title bar of the `Command window`. For example if the Main Window currently has the focus, the `Command window` would appear as:

VALUES A valid reference to a visible viewer object.

font

Resource Type

DESCRIPTION The resource type name for a `font` resource.

Font resources are referenced by `name` or `ID` number with the `font` keyword.

EXAMPLES

```
installFontResource font "Arial"
```

fontFace

Property

DESCRIPTION A button, combobox, field, or recordfield property that specifies the font for button labels and combobox, field, or recordfield text.

NOTES You can get or set this property.

If the value for `fontFace` is a font that is not available on the user's system, ToolBook displays the text in an available font.

If the book is later opened on another system that has the specified font, the text displays correctly.

VALUES The name of the font for the object's text.

The default is the value of `sysFontFace` when the object was created.

EXAMPLES

```
fontFace of button "start" = "ms sans serif"
fontFace of field "entry" = "Arial"
```

fontSize

Property

DESCRIPTION A button, combobox, field, or recordfield property that specifies the point size of button labels and combobox, field, or recordfield text.

NOTES You can get or set this property.

If the value specified for `fontSize` is not available on the user's system, the next point size smaller is used.

If you are using a True Type font, then this is not a concern since True Type fonts can scale to any desired size.

VALUES A positive whole number that defines the point size of the font face for the selected object; the default is the value of `sysFontSize` when the object was created.

EXAMPLES
```
-- Magnify text is a field by increasing its size
increment fontSize of field "info" by 2
```

fontStyle

Property

DESCRIPTION A button, combobox, field, or recordfield property that specifies the font style of button labels and combobox, field, or recordfield text.

NOTES You can get or set this property.

You can format the color, font face, style, and size of individual characters in a field, recordfield, or combobox edit field.

To change a character's style, select the text, then send the menu event message for the desired style.

For example:
```
select chars 2 to 133 of text of field "house listings"
send italic
```

To remove all values for an object's `fontStyle`, set its value to regular (or `null`).

VALUES `Regular` (or `null`), or a list of one or more of the values `bold`, `italic`, `underline`, `strikeout`, `superscript`, or `subscript`

The default is the value of `sysFontStyle` when the object was created.

EXAMPLES
```
fontStyle of field "jokes" = "bold,underline"

fontStyle of button "start" = null

select textlines 1 to 5 of text of field "listing"
send underline
```

footer

Property

DESCRIPTION A book property that specifies the contents of a footer to be printed on each sheet of a report.

NOTES You can get or set this property.

VALUES A string of up to `32,000` characters.

The default is `null` (no footer).

To specify a `print time`, `print date`, or `running page number`, enter `~t`, `~d`, or `~p`, respectively.

The font of the footer is determined by the values for `sysFontFace`, `sysFontSize`, and `sysFontStyle` when the footer is set.

EXAMPLES
```
footer of this book = "Draft Date: ~d      Page: ~p"
```

for

Articles and Prepositions

DESCRIPTION Used in conjunction with various OpenScript terms. This includes the following: `readFile`, `search`, and `seekFile`.

EXAMPLES

```
search page for searchText

readFile myFile for 50

seekFile "list.txt" for -230 from end
```

foreground

Working Layer

DESCRIPTION Sent to the page when you choose `Foreground` from the `View` menu.

You can also `send` this message using the `send foreground` statement. ToolBook's default response is to switch the working layer from the background to the foreground (page layer) of the current page.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
-- Adds a recordfield to a book
to handle addRecordField pLoc1, pLoc2, pName
    sysLockScreen = true
    send background
    draw recordfield from pLoc1 to pLoc2
    name of selection = pName
    send foreground
    sysLockScreen = false
end
```

format

Data Manipulation

SYNTAX `format [<type>] <container> [as <new format>] [from <old format>]`

DESCRIPTION Converts numbers, dates, or times in a container to a specified format. The most common uses of the `format` command are to format numbers, dates, and times for display or for calculations:

- Numbers as `sysNumberFormat` for display
- Numbers as `null` for calculations
- Numbers as `@d`, `@h`, `@o`, and `@b` for decimal, hexadecimal, octal, and binary number formats
- Dates and times as `sysDateFormat` or `sysTimeFormat` for display
- Dates and times as `seconds` for calculations

You can also use the `format` command to set up character-by-character formats for displaying numbers, dates, and times.

NUMERIC FORMATS

ToolBook uses special number format symbols for calculations, general precision formatting, and decimal, hexadecimal, octal, and binary numbers. The following symbols represent these special formats:

`null`	calculation	`@h or @H`	hexadecimal (case-sensitive)
`"?"`	general precision	`@o`	octal
`@d`	decimal	`@b`	binary

- Use `null` primarily to format numbers for calculations. ToolBook displays the number as precisely as possible, using scientific notation when necessary. When a number is formatted as `null`, ToolBook strips out leading and trailing spaces, percent symbols (`%`), and the `sysCurrency` and `sysThousand` characters, and it converts the `sysDecimal` character to a period.

- "`?`" is the symbol for general precision format and is the default value of `sysNumberFormat`. ToolBook displays the number as precisely as possible, and also correctly interprets the characters for the `sysCurrency`, `sysDecimal`, and `sysThousand` characters that are embedded in the number.

For calculations, format numbers as `null`. For display, format numbers as "`?`", `sysNumberFormat`, or a string of placeholder characters enclosed in quotation marks.

Numbers that use only the characters `0` through `9` and a period for the decimal separator are valid for calculations. If the numbers for calculations are displayed or imported using these characters, it is not necessary to execute `format number` before performing a calculation. Note that numbers used in calculations cannot contain commas.

ToolBook recognizes decimal, hexadecimal, octal, and binary formatting symbols only if they are the first characters of the format string. ToolBook interprets these symbols literally if they are anywhere but at the beginning of the string.

The hexadecimal formatting symbol is case-sensitive, so specifying `@H` or `@h` produces different output. For example:

```
x = 26
format number x as "@h0"
request x                --Returns "1a"
format number x as "@H0"
request x                --Returns "1A"
```

To use the "`@`" symbol as the first character of a formatting string, use `@@`.

PLACEHOLDER SYMBOLS FOR NUMBER FORMATS

SYMBOL	MEANING
`Literal characters`	Character that ToolBook interprets literally. For example, if the format is "`IT is 0`" and the number is 2, ToolBook displays "`IT is 2`". Some characters can cause unexpected results. For example, `d` is interpreted as a date format as described in the table "Placeholder symbols for date and time formats" later in this entry. Also, when a number is reformatted from a format using literal characters to the `null` format, ToolBook can't strip out the literal characters, so the number might not be valid for calculations.
`sysCurrency character`	Value of `sysCurrency`. Used like a literal character in a placeholder string, except that ToolBook can strip out this character when the number is formatted as `null`.
`sysThousand character`	Value of `sysThousand`. Used like a literal character in a placeholder string, except that ToolBook can strip out this character when the number is formatted as `null`.
`sysDecimal character`	Value of `sysDecimal`. Used like a literal character in a placeholder string, except that ToolBook converts this character to a period when the number is formatted as `null`.
`0`	ToolBook displays a zero if there are fewer digits in a number than places in the format. If there are more digits than places, ToolBook displays all digits to the left of the decimal point and rounds digits to the right. For example, if the format is "`000.00`" for the numbers `34` and `3457.888`, ToolBook displays `034.00` and `3457.89`.

SYMBOL	MEANING
#	Same as 0, except ToolBook displays a space if there are fewer digits in a number than places in the format. For example, if the format is "###.##" for the number 34, ToolBook displays a space, then 34.
E- or e-	ToolBook displays numbers in scientific notation if one of these placeholders appears to the left of a 0 or # placeholder, which determines the number of digits in the exponent. Negative exponents are displayed with a minus sign, and positive exponents are displayed without a sign.
E+ or e+	Same as E- and e-, except that in this format ToolBook displays positive exponents with a plus sign.

DATE AND TIME FORMATS

For calculations, format dates and times as seconds. For display, format dates and times as sysDateFormat or sysTimeFormat, or with a string of placeholders that is enclosed in quotation marks if it includes spaces or characters other than letters and numbers. To format dates and times as lists, use a comma as the separator character between month, day, year, hour, and minutes.

PLACEHOLDER SYMBOLS FOR DATE AND TIME FORMATS

SYMBOL	MEANING
M	Month's name (January ... December).
MMM	Month's three letter abbreviation (Jan ... Dec).
m	Month's number (1 ... 12).
mm	Month's number, leading 0 for 1 through 9 (01 ... 12).
d	Day's number (1 ... 31).
dd	Day's number, leading 0 for 1 through 9 (01 ... 31).
y	Year's number as an integer.
yy	Year's last two digits.
h	Hour on a 12-hour clock (1 ... 12).
hh	Hour on a 12-hour clock, leading 0 for single digits (01 ... 12).
h24	Hour on a 24-hour clock (0 ... 23).
hh24	Hour on a 24-hour clock, leading 0 for single digits (00 ... 23).
min	Minute (00 ... 59).
sec	Second (00 ... 59).
AMPM	AM for morning, PM for afternoon.
seconds	For date arithmetic, seconds elapsed since 00:00:00 GMT (Greenwich mean time) on January 1, 1970. For time arithmetic, seconds elapsed since 00:00:00 of the current day. Note that the seconds format is valid for dates through January 18, 2038.

If a field contains a numeric value and text, use a string specifier to refer to the number for the calculation. Don't refer to the text of the entire field because ToolBook won't recognize the letters and other non-numeric characters as valid for the calculation.

Use decimal tab spacing or a monospace font, such as Courier, to ensure alignment of numbers in columns. Proportionally spaced fonts, such as Arial and Times New Roman, can result in uneven spacing for numeric formats.

NOTES When formatting a number with a placeholder string, avoid using characters other than digits, a percent symbol, spaces, and the values of sysCurrency, `sysThousand`, and `sysDecimal`. ToolBook can strip these characters when a number is formatted to `null` for a calculation. ToolBook may not be able to strip other letters and special characters; therefore, the value might not be valid for a numeric calculation.

PARAMETERS

PARAMETER	DESCRIPTION
`<type>`	`Number`, `date`, or `time`; the default is `number`.
`<container>`	A container or a string specifier indicating the characters to format. To format fields, recordfields, or comboboxes, refer to the `text` property of the object. Or select specific characters and format the `selectedText` system property.
`<new format>` `<old format>`	For numbers: `null`, "?", `sysNumberFormat`, or a string of placeholders enclosed in quotation marks that define a character-by-character format. For dates and times: `sysDateFormat`, `sysNumberFormat`, or a string of placeholders enclosed in quotation marks that define a character-by-character format. To format a container as `null` from the `sysNumberFormat`, omit the `<old format>` and `<new format>` parameters; for example, `format number text of field "Total"`. To format a container as the current value of `sysNumberFormat`, `sysDateFormat`, or `sysTimeFormat`, omit the `<new format>` parameter; for example, `format number text of field "Total" from null`.

EXAMPLES

```
format time last textline of footer of this book from "hh24:min:sec"

format date textline 1 of text of field "Date" \
    as "y MMM dd"  from "mm/dd/yy"

format text of field "Quantity" as "#.00"

to handle displaytime
    local vTime
    put sysTime into vTime
    format time vTime as "hh:min"
    put "The time now is" && vTime
end

--Creates binary number formatting for data in a variable
vData = 10
format vData as "@b#"
request vData          --Displays "1010" in the Request dialog box
```

formfeed

Spacing Constants

DESCRIPTION Represents the formfeed `ANSI` character. This is equivalent to `ANSI 12`.

EXAMPLES

```
fileText = part_1 & formfeed & part_2
fName = "c:\law.txt"
createFile fName
writeFile fileText to fName
closeFile fName
```

forth

Ordinal Numbers

DESCRIPTION A value used to indicate the forth item in a sequence or list. Also used to indicate relative position of pages or backgrounds.

EXAMPLES

```
-- The following paired examples show how you can use ordinal
-- references to make OpenScript easier to read. As you can see it is
-- possible to write your scripts with or without ordinal references.
go to the forth page of this book
go to page 4 of this book

if forth character of str = "X"
if character 4 of str = "X"

forth word of text of field "lesson" = "Hello"
word 4 of text of field "lesson" = "Hello"

put "Cancelled:" into forth character of name of button id 16
put "Cancelled:" into character 4 of name of button id 16
```

forward

Program Flow

SYNTAX

```
forward
forward <message> [<parameters>]
forward to system
forward to parent
```

DESCRIPTION Sends a message to the next highest object in the object hierarchy or directly to the ToolBook system.

If a `message` is not specified, the message currently being handled and its parameters are sent to the object.

The `forward to system` form sends the current message and its parameters directly to the ToolBook system.

The calling handler regains control after any called handlers are executed.

NOTES When you work with system books, the use of the `forward` command is very important. For example, if a handler for the same message is in the script of a background and also in the script of the system book, you must place a forward statement for the message in the background script. Otherwise the handler in the system book will never be called.

You can use the `forward <message>` statement to send a message up the object hierarchy, starting with the parent of the current object.

When used in a `to get` handler, `forward` sets the value of `IT` to the value returned from `to get` handlers that are higher in the object hierarchy.

When you use `forward to system` to forward a translated Window message to the system, ToolBook calls the default Windows procedure and puts the value returned from the procedure into `IT`.

If you use a handler for the `menuItemSelected` message in a viewer or a book, be sure to include a `forward` or `forward to system` statement in the handler to allow ToolBook to send `menu event` messages.

When you include a `forward` in a `shared script`, the hierarchy is: `objectScript`, `objectSharedScript`, `parentObjectScript`, then `parentObjectSharedScript`. If forwarding is required and you include a normal `forward` in the `SharedScript`, you will encounter an infinite loop. Instead, use `forward to parent` to force the continuation to the next level.

If you `forward` a message from a viewer, the message is sent to the book that owns the viewer.

PARAMETERS

PARAMETER	DESCRIPTION
<message>	A string that represents a built-in or user-defined message.
<parameters>	An optional list of parameters for the message.

EXAMPLES

```
to handle mouseEnter
    forward
    text of field "time" = systime
end
```

frameToPageUnits()

Screen Display Function

SYNTAX

```
frameToPageUnits(<coordinates>,<viewer reference>)
```

DESCRIPTION

Converts `pixels` relative to the `client area` of a specified viewer into `page units`.

This function accounts for the current values of the viewer's `pageScroll` and `magnification` properties.

Use this function to align an object on a page with a tiled viewer (a tiled viewer is a child of its parent viewer's frame window).

NOTES

The conversion of frame-relative `pixels` to `page units` is dependent upon the currently installed video driver.

PARAMETERS

PARAMETER	DESCRIPTION
<coordinates>	A list of numbers in `pixels` that specify a position or positions on the screen. You can include as many numbers as necessary, but they must be in pairs. You can refer to the `bounds`, `position`, `size`, or `vertices` property as the value for <coordinates>.
<viewer reference>	A valid viewer reference. If the specified viewer is not open, ToolBook displays an error message.

frameToScreen()

Screen Display Function

SYNTAX

```
frameToScreen(<coordinates>,<viewer reference>)
```

DESCRIPTION

Converts `pixels` relative to the client area of a specified viewer's frame window into `pixels` relative to the origin of the screen.

NOTES

Use this function to align a popup window with a tiled viewer (a tiled viewer is a child of its parent viewer's frame window).

PARAMETERS

PARAMETER	DESCRIPTION
<coordinates>	A list of numbers in `pixels` that specify a position or positions on the screen. You can include as many numbers as necessary, but they must be in pairs. You can refer to the `bounds`, `position`, `size`, or `vertices` property as the value for <coordinates>.
<viewer reference>	A valid viewer reference. If the specified viewer is not open, ToolBook displays an error message.

from

Articles and Prepositions

DESCRIPTION Used in conjunction with various OpenScript terms. This includes the following: `draw`, `format`, `select all`, and `step`.

EXAMPLES
```
draw ellipse from 0,0 to 300,600
step k from 1 to x by 2
select all from 20,30 to 600,900
```

fv()

Financial Values

SYNTAX `fv(<rate>,<nper>,<payment>,<present value>[,<type>])`

DESCRIPTION A financial function that returns the future value of an investment. This function is based on periodic, constant payments and a constant interest rate. Enter p ayments out of pocket as negative values, and enter income as positive values.

PARAMETERS

PARAMETER	DESCRIPTION
`<rate>`	A positive whole number representing the interest rate per period.
`<nper>`	A positive whole number representing the total number of pay ment periods in an annuity.
`<payment>`	A number representing the payment made each period; it must remain constant over the life of an annuity. Typically, `<payment>` contains principal and interest but no other fees or taxes.
`<present value>`	A number representing the present value, or the lump sum amount that a series of future payments is worth at the present time. If `<present value>` is omitted, it is assumed to be `0`.
`<type>`	Optional Parameter. A value of `true` or `false` that indicates when payments are due. `True` indicates that payments are due at the beginning of the period (annuity due), `false` at the end (ordinary annuity). If `<type>` is omitted, it is assumed to be `false`.

EXAMPLES
```
-- Calculates the future value of an investment in ten years
x = fv(0.095,10,-650,0,false)
put x into text of field "future value"
```

fxDissolve

Transition Effects

SYNTAX `fxDissolve [<speed>] [to <result>]`

DESCRIPTION A visual effect that appears to dissolve the current page to reveal another image.

PARAMETERS

PARAMETER	DESCRIPTION
`<speed>`	`Slow`, `normal`, or `fast`; the default is `normal`.
`<result>`	`Black`, `gray`, `white`, or the unique name of a page; the default is `black`.

EXAMPLES
```
to handle buttonClick
    fxDissolve -- transitions to a black screen
    go page "help"
end

to handle buttonClick
    fxDissolve slow to page "help"
end
```

fxWipe

SYNTAX `fxWipe <direction> [<speed>] [to <result>]`

DESCRIPTION A visual effect that appears to wipe the current page off the screen to reveal another image.

NOTES The motion starts from the center of the page or from a specified point and moves outward.

PARAMETERS

PARAMETER	DESCRIPTION
<direction>	`Top`, `bottom`, `left`, or `right`, indicating which edge of the screen the wipe moves toward.
<speed>	`Slow`, `normal`, or `fast`; the default is `normal`.
<result>	`Black`, `gray`, `white`, or the unique name for a page; the default is `black`.

EXAMPLES

```
to handle buttonClick
    fxWipe left -- transitions to a black screen
    go page "help"
end

to handle buttonClick
    fxWipe right fast to page "help"
end
```


fxZoom

SYNTAX `fxZoom [<speed>] [to <result>] [at <location>]`

DESCRIPTION A visual effect that appears to wipe the current page in an enlarging rectangle to reveal another image.

NOTES The motion starts from the center of the page or from a specified point and moves outward.

PARAMETERS

PARAMETER	DESCRIPTION
<speed>	`Slow`, `normal`, or `fast`; the default is `normal`.
<result>	`Black`, `gray`, `white`, or the unique name for a page; the default is `black`.
<location>	A list of two numbers in `page units`, where the first number is the distance from the window's left edge and the second is the distance from the top of the window. The default is the center of the displayed area of the page.

EXAMPLES

```
to handle buttonClick
    fxZoom -- transitions to a black screen
    go page "help"
end

to handle buttonClick
    fxZoom fast to page "help" at 500,500
end
```

get

Data Manipulation

SYNTAX `get <expression>`

DESCRIPTION Evaluates an expression and places the result into the local variable, `IT`.

NOTES If you attempt to use the `get` command with a property name that is not built into ToolBook (for example, `get mySpecialProperty of button "help"`), ToolBook will search the object hierarchy for a corresponding `to get` handler for the user property name. If no such handler exists, ToolBook treats the name as a user property and attempts to get its value.

When you get the value of a built-in ToolBook property, ToolBook does not search for a corresponding `to get` handler for its name.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	Any expression that yields a value.

EXAMPLES

```
get sysTime
request IT
```

getCDDriveList()

TBDOS.DLL File Function (Drive)

SYNTAX `getCDDriveList()`

DESCRIPTION Returns a string that specifies a list of CD drives on the current system.

RETURNS The list of drive letters is returned as a `CRLF`-separated list.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    STRING getCDDriveList()
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getCDDriveList32()` instead.

PARAMETERS No Parameters.

EXAMPLES

```
to handle buttonClick
    dvs = getCDDriveList()
    if textlineCount(dvs) > 1
        -- user has more than one CD Drive on their system
        firstDrive = textline 1 of dvs
        request "This application will require that you place the " & \
                "CD ROM into your " & firstDrive & " drive."
    end
end
```

getCDDriveList32()

TBFILE32.DLL File Function (Drive)

SYNTAX

```
getCDDriveList32()
```

DESCRIPTION

Returns a string that specifies a list of CD drives on the current system.

RETURNS

The list of drive letters is returned as a `CRLF`-separated list.

NOTES

As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    STRING getCDDriveList32()
end
```

PARAMETERS

No Parameters.

EXAMPLES

```
to handle buttonClick
    dvs = getCDDriveList32()
    if textlineCount(dvs) > 1
        -- user has more than one CD Drive on their system
        firstDrive = textline 1 of dvs
        request "This application will require that you place the " & \
                "CD ROM into your " & firstDrive & " drive."
    end
end
```

getCurrentDirectory()

TBDOS.DLL File Function (Directory)

SYNTAX

```
getCurrentDirectory(<drive letter>)
```

DESCRIPTION

Gets the current working directory for the specified disk drive.

This function will return short file names only. Use `getCurrentDirectoryLFN()` if you need to return long file names.

RETURNS

If no error occurs, `getCurrentDirectory()` returns a string that contains the path of the current working directory.

Otherwise, `sysError` is set to:

```
 -1  Internal error occurred.
-20  Memory allocation error.
```

NOTES

You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    STRING getCurrentDirectory(STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getCurrentDirectory32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<drive letter>`	The letter that designates the drive.

EXAMPLES

```
cd = getCurrentDirectory("c")
```

getCurrentDirectory32()

TBFILE32.DLL File Function (Directory)

SYNTAX `getCurrentDirectory32(<drive letter>)`

DESCRIPTION Gets the current working directory for the specified disk drive.

RETURNS If no error occurs, `getCurrentDirectory32()` returns a string that contains the path of the current working directory.

Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    STRING getCurrentDirectory32(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<drive letter>`	The letter that designates the drive.

EXAMPLES

```
get setCurrentDirectory32("c:\tb\training")
```

getCurrentDirectoryLFN()

TBDOS.DLL File Function (Directory)

SYNTAX `getCurrentDirectoryLFN(<drive letter>)`

DESCRIPTION Gets the current working directory for the specified disk drive.

RETURNS If no error occurs, `getCurrentDirectoryLFN()` returns a string that contains the path of the current working directory.

Otherwise, `sysError` is set to:

`-1` Internal error occurred.
`-20` Memory allocation error.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    STRING getCurrentDirectoryLFN(STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getCurrentDirectory32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<drive letter>`	The letter that designates the drive.

EXAMPLES

```
cd = getCurrentDirectoryLFN("c")
```

getCurrentDrive()

TBDOS.DLL File Function (Drive)

SYNTAX `getCurrentDrive()`

DESCRIPTION Gets the current disk drive letter.

RETURNS If no error occurs, `getCurrentDrive()` returns a one-character string containing the current disk drive letter.

Otherwise, the function returns `null` and sets `sysError` to a negative value.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    STRING getCurrentDrive()
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getCurrentDrive32()` instead.

PARAMETERS No parameters.

EXAMPLES

```
curDrive = getCurrentDrive()
```

getCurrentDrive32()

TBFILE32.DLL File Function (Drive)

SYNTAX `getCurrentDrive32()`

DESCRIPTION Gets the current disk drive letter.

RETURNS If no error occurs, `getCurrentDrive32()` returns a one-character string containing the current disk drive letter.

Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    INT STRING getCurrentDrive32()
end
```

PARAMETERS No parameters.

EXAMPLES

```
curDrive = getCurrentDrive32()
```

getCustomColors()

TBDLG.DLL Color Functions

SYNTAX `getCustomColors()`

DESCRIPTION Returns a `CRLF`-delimited list of the `16` colors that represent the current setting for the custom color palette used by the `colorPaletteDlg()` function.

Each color is represented by a list of three numbers that specify one `RGB` color.

RETURNS If no error occurs, `getCustomColors()` returns a list of colors and `sysError` is set to `1`.

If the system is unable to allocate enough memory to create the string, the function returns `null` and `sysError` is set to `-2`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING getCustomColors()
end
```

PARAMETERS No parameters

EXAMPLES

```
cls = getCustomColors()
```

getDirectoryOnlyList()

TBDOS.DLL File Function (Directory)

SYNTAX

```
getDirectoryOnlyList(<path>,<sort order>)
```

DESCRIPTION Returns a list of matching files based on the specified path and file name. The function will return short file names only. Use `getDirectoryOnlyListLFN()` if you need to return long file names.

RETURNS If no error occurs, `getDirectoryOnlyList()` returns a `CRLF`-separated list of files. One file appears on each line.

If an error occurs, the function returns `null` and `sysError` is set to one of these values:

```
 -2  Specified path was invalid.
 -3  Specified file attribute was invalid.
-20  Memory allocation error.
```

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    STRING getDirectoryOnlyList(STRING,STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getDirectoryOnlyList32 ()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<path>`	A string that contains the name of the path and file you want. The string can contain `?` and `*` wildcards.
`<sort order>`	A string containing a character that specifies the sort order for the returned list. The values for `<sort order>` use the same characters as the DOS `DIR` command. You can use one of the following values: `N =` by name `D =` by date `E =` by extension `NULL(0) or "" =` unsorted `S =` by size If the `<sort order>` parameter is preceded by a hyphen (`-`), the sort order is reversed.

EXAMPLES

```
dirs = getDirectoryOnlyList("c:\","N")
```

getDirectoryOnlyList32()

TBFILE32.DLL File Function (Directory)

SYNTAX `getDirectoryOnlyList32(<path>,<sort order>)`

DESCRIPTION Returns a list of matching files based on the specified path and file name.

RETURNS If no error occurs, `getDirectoryOnlyList32()` returns a `CRLF`-separated list of files. One file appears on each line.

Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 `getFileAttributes32()` is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    STRING getDirectoryOnlyList32(STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<path>`	A string that contains the name of the path and file you want. The string can contain `?` and `*` wildcards.
`<sort order>`	A string containing a character that specifies the sort order for the returned list. The values for `<sort order>` use the same characters as the DOS `DIR` command. You can use one of the following values: `N =` by name `D =` by date `E =` by extension `NULL(0) or "" =` unsorted `S =` by size If the `<sort order>` parameter is preceded by a hyphen (`-`), the sort order is reversed.

EXAMPLES

```
dirs = getDirectoryOnlyList32("c:\","N")
```

getDirectoryOnlyListLFN()

TBDOS.DLL File Function (Directory)

SYNTAX `getDirectoryOnlyListLFN(<path>,<sort order>)`

DESCRIPTION Returns a list of matching files based on the specified path and file name.

RETURNS If no error occurs, `getDirectoryOnlyListLFN()` returns a `CRLF`-separated list of files. One file appears on each line.

If an error occurs, the function returns `null` and `sysError` is set to one of these values:

`-2` Specified path was invalid.
`-3` Specified file attribute was invalid.
`-20` Memory allocation error.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    STRING getDirectoryOnlyListLFN(STRING,STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getDirectoryOnlyList32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
<path>	A string that contains the name of the path and file you want. The string can contain ? and * wildcards.
<sort order>	A string containing a character that specifies the sort order for the returned list. The values for <sort order> use the same characters as the DOS DIR command. You can use one of the following values: N = by name D = by date E = by extension NULL(0) or "" = unsorted S = by size If the <sort order> parameter is preceded by a hyphen (-), the sort order is reversed.

EXAMPLES

```
dirs = getDirectoryOnlyListLFN("c:\,"N")
```

getDriveKind()

TBDOS.DLL File Function (Drive)

SYNTAX

```
getDriveKind(<drive letter>)
```

DESCRIPTION

Returns the type of the specified drive.

RETURNS

If no error occurs, `getDriveKind()` returns a string containing one of these values:

VALUE	DESCRIPTION
removable	A removable drive such as a floppy drive or zip drive.
fixed	A non removable drive - such as your main hard drive.
CD-ROM	A CD ROM Drive or a DVD Drive.
network	A network location.

Otherwise, the function returns `null` and sets `sysError` to one of these values:

-1 Invalid drive letter.
-20 Memory allocation error.

NOTES

You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    STRING getDriveKind(STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getDriveKind32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
<drive letter>	A string that contains the letter of the drive. The string can be an entire path but must specify a letter as the first character.

EXAMPLES

```
-- Display a list of current drive letters and drive types
to handle buttonClick
    dvs = getDriveList()
    results = null
    step k from 1 to textlineCount(dvs)
        curDrive = textline k of dvs
        curDriveType = getDriveKind(curDrive)
        textline k of results = curDrive && ":" && curDriveType
    end
    request results
end
```

getDriveKind32()

TBFILE32.DLL File Function (Drive)

SYNTAX `getDriveKind32(<drive letter>)`

DESCRIPTION Returns the type of the specified drive.

RETURNS If no error occurs, `getDriveKind32()` returns a string containing one of these values:

VALUE	DESCRIPTION
`removable`	A removable drive such as a floppy drive or zip drive.
`fixed`	A non removable drive - such as your main hard drive.
`CD-ROM`	A CD ROM Drive or a DVD Drive.
`network`	A network location.
`RAM`	A RAM Disk (memory configure to act as a drive)

Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    STRING getDriveList32()
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<drive letter>`	The letter designating the disk drive.

EXAMPLES

```
-- Display a list of current drive letters and drive types
to handle buttonClick
    dvs = getDriveList32()
    results = null
    step k from 1 to textlineCount(dvs)
        curDrive = textline k of dvs
        curDriveType = getDriveKind32(curDrive)
        textline k of results = curDrive && ":" && curDriveType
    end
    request results
end
```

getDriveList()

TBDOS.DLL File Function (Drive)

SYNTAX `getDriveList()`

DESCRIPTION Gets a list of valid drives on the current machine.

RETURNS If no error occurs, `getDriveList()` returns a list of drives for the current machine, separated by `CRLFs`.

If not enough memory is available to build the list, the function returns `null` and sets `sysError` to `-20`.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    STRING getDriveList()
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getDriveList32()` instead.

PARAMETERS No parameters.

EXAMPLES

```
-- Display a list of current drive letters and drive types
to handle buttonClick
    dvs = getDriveList()
    results = null
    step k from 1 to textlineCount(dvs)
        curDrive = textline k of dvs
        curDriveType = getDriveKind(curDrive)
        textline k of results = curDrive && ":" && curDriveType
    end
    request results
end
```

getDriveList32()

TBFILE32.DLL File Function (Drive)

SYNTAX

```
getDriveList32()
```

DESCRIPTION Gets a list of valid drives on the current machine.

RETURNS If no error occurs, `getDriveList32()` returns a list of drives for the current machine, separated by `CRLFs`.

Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    STRING getDriveList32()
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<drive letter>`	The letter designating the disk drive.

EXAMPLES

```
-- Display a list of current drive letters and drive types
to handle buttonClick
    dvs = getDriveList32()
    results = null
    step k from 1 to textlineCount(dvs)
        curDrive = textline k of dvs
        curDriveType = getDriveKind32(curDrive)
        textline k of results = curDrive && ":" && curDriveType
    end
    request results
end
```

getEllipsisByCharCount32()

TBFILE32.DLL String Functions

SYNTAX

```
getEllipsisByCharCount32(<chars>,<fileRef>,<nFileRef>,<nPath>)
```

DESCRIPTION This function will modify a string containing a file reference, by reducing the size of the string to the number characters specified. This is done by ellipsizing the string.

Use this function when you want to display a file reference where limited space has been provided.

For example:

```
c:\book\chapter\page\figures\subheading\text.doc
c:\...\subheading\text.doc
```

RETURNS The returned value will contain the ellipsized file reference.

The original file reference will be returned unmodified if it is already equal to or less than the value specified in `<chars>`.

If there is an error, `null` is returned and `sysError` is set to `-1`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    STRING getEllipsisByCharCount32(INT,STRING,INT,INT)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<chars>`	Maximum number of characters in the resulting string.
`<fileRef>`	The file to ellipsize. Can be just a file name or a filename with pathing information.
`<nFileRef>`	`1` Returns the entire filename. `0` Implies that the filename may be ellipsized, if it exceeds the maximum width.
`<nPath>`	`1` Shows entire subdirectory names. `0` Implies that a subdirectory may be ellipsized.

NOTES

`fileSpec = c:\abcdefg\bcdefgh\cdefghi\defghij\thisIsAlongFileName.tbk`

`<nFileRef>` If `TRUE`, always returns the entire filename, even if it exceeds the max value in `<chars>`. If `FALSE`, ellipsizes the end of the filename, if the filename is too large.

`<nPath>` If `TRUE`, subdirectories are removed in their entirety. If `FALSE`, a subdirectory may be ellipsized.

`case #1` Using the example `fileSpec` above, where the maximum number of characters allowed is 18:

`<nFileRef> = 1` returns "thisIsAlongFileName.tbk", which is 23 characters.

`<nFileRef> = 0` returns "thisIsAlongFile…" which is 18 characters.

Since the filename exceeds the max, `<nPath>` will be ignored regardless of its setting.

`case#2` Using the example `fileSpec` above, if the maximum number of characters is 48:

`<nPath> = 0` returns "c:\...gh\cdefghi\defghij\thisIsAlongFileName.tbk" which is 48 characters.

`<nPath> = 1` returns "c:\...\defghij\thisIsAlongFileName.tbk" which is 38 characters.

Since the filename < the max implies at least the entire filename is always safely returned, `<nPath>` is not ignored, but `<nPath>` is.

EXAMPLES

```
to handle buttonClick
    dir = sysToolBookDirectory
    fName = "tbfile32.dll"
    fullPath = dir & fName
    request fullPath
    shortPath = getEllipsisByCharCount32(32,fullPath,1,1)
    request shortPath
end
```

SYNTAX

```
GetEllipsisByFont32(font>,<size>,<bold>,<italic>,<fileRef>,<width>,
  <nFileRef>,<nPath>)
```

DESCRIPTION

This function will modify a string containing a file reference, by reducing the length of the string to the number characters which will fit within the width specified. This is done by analyzing the font characteristics you specify.

A use for this function is when you want to display a file reference in a field where the ellipsized file reference will not exceed the width of the field.

For example:

```
c:\book\chapter\page\figures\subheading\text.doc
c:\...\subheading\text.doc
```

RETURNS

The returned value will contain the ellipsized file reference.

The original file reference will be returned unmodified if it is already equal to or less than the value specified in `<chars>`.

If there is an error, `null` is returned and `sysError` is set to `-1`.

NOTES

As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    STRING getEllipsisByFont32(STRING,INT,INT,INT,STRING,INT,INT,INT)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<font>`	The name of the font you will be using to display your string.
`<size>`	The point size of the font you will be using to display your string.
`<bold>`	`1` Used if the string you will be displaying will be Bold. `0` Used if the string you will be displaying will not be Bold.
`<italic>`	`1` Used if the string you will be displaying will be Italic. `0` Used if the string you will be displaying will not be Italic.
`<fileRef>`	The file to ellipsize. Can be just a file name or a filename with pathing information.
`<width>`	The maximum with in `page units` of the return string.
`<nFileRef>`	`1` Returns the entire filename. `0` Implies that the filename may be ellipsized, if it exceeds the maximum width.
`<nPath>`	`1` Shows entire subdirectory names. `0` Implies that a subdirectory may be ellipsized.

NOTES

```
fileSpec = c:\abcdefg\bcdefgh\cdefghi\defghij\thisIsAlongFileName.tbk
```

`<nFileRef>` If `TRUE`, always returns the entire filename, even if it exceeds the max value in `<chars>`. If `FALSE`, ellipsizes the end of the filename, if the filename is too large.

`<nPath>` If `TRUE`, subdirectories are removed in their entirety. If `FALSE`, a subdirectory may be ellipsized.

`case #1` Using the example `fileSpec` above, where the maximum number of characters allowed is 18:

`<nFileRef> = 1` returns "thisIsAlongFileName.tbk", which is 23 characters.

`<nFileRef> = 0` returns "thisIsAlongFile…" which is 18 characters.

Since the filename exceeds the max, `<nPath>` will be ignored regardless of its setting.

`case#2` Using the example `fileSpec` above, if the maximum number of characters is 48:

`<nPath> = 0` returns "c:\...gh\cdefghi\defghij\thisIsAlongFileName.tbk" which is 48 characters.

`<nPath> = 1` returns "c:\...\defghij\thisIsAlongFileName.tbk" which is 38 characters.

Since the filename < the max implies at least the entire filename is always safely returned, `<nPath>` is not ignored, but `nPath` is.

EXAMPLES

```
to handle buttonClick
    dir = sysToolBookDirectory
    fName = "tbfile32.dll"
    fullPath = dir & fName
    request fullPath
    shortPath = getEllipsisByCharCount32(32,fullPath,1,1)
    request shortPath
end
```

getFileAttributes()

TBDOS.DLL File Function (Attribute)

SYNTAX `getFileAttributes(<file name>)`

DESCRIPTION Gets the file attributes of the specified file.

RETURNS If no error occurs, `getFileAttributes()` returns a string containing a character for each attribute that is true:

```
R = read-only     D = directory
H = hidden        V = volume label
S = system        A = archive
```

If an error occurs, the function returns `null` and `sysError` is set to one of these values:

`-2` Specified file was not found.
`-3` Specified path was invalid.
`-5` Access to the file was denied.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    STRING getFileAttributes(STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getFileAttributes32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
<file name>	The name of the file with the attributes you want.

EXAMPLES

```
if getFileAttributes(fName) contains "r"
    request "Sorry, file is read-only."
end
```

getFileAttributes32()

TBFILE32.DLL File Function (Attribute)

SYNTAX

```
getFileAttributes32(<file name>)
```

DESCRIPTION

Gets the file attributes of the specified file.

RETURNS

If no error occurs, `getFileAttributes()` returns a string containing a character for each attribute that is true:

```
R = read-only     D = directory
H = hidden        V = volume label
S = system        A = archive
```

Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES

As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    STRING getFileAttributes32(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<file name>	The name of the file with the attributes you want.

EXAMPLES

```
if getFileAttributes32(fName) contains "r"
    request "Sorry, file is read-only."
end
```

getFileDate()

TBDOS.DLL File Function (Attribute)

SYNTAX

```
getFileDate(<file name>)
```

DESCRIPTION

Gets the date and time for the specified file.

Returns a four digit number for the year if the year is `2000` or higher, or a two digit number for the year `1999` or earlier.

RETURNS

If no error occurs, `getFileDate()` returns a string containing the file's date and time.

If there is an error, `sysError` is set to one of these values:

`-2` Specified file was not found.
`-3` Specified path was invalid.
`-20` Memory allocation error.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    STRING getFileDate(STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getFileDate32()` instead.

The `ASYM_GetFileDate()` function is similar to `getFileDate()`, but it handles localization and dates beyond the year `2000` more efficiently, because you can specify the format of the return value.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	The name of the file for which you want the date.

EXAMPLES

```
x = getFileDate("c:\boot.txt")
```

getFileDate32()

TBFILE32.DLL File Function (Attribute)

SYNTAX

```
getFileDate32(<file name>)
```

DESCRIPTION Gets the date and time for the specified file.

Returns a four digit number for the year if the year is `2000` or higher, or a two digit number for the year `1999` or earlier.

RETURNS If no error occurs, `getFileDate32()` returns a string containing the file's date and time.

If there is an error, `sysError` is set to one of these values:

`-2` Specified file was not found.
`-3` Specified path was invalid.
`-20` Memory allocation error.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    STRING getFileDate32(STRING)
end
```

The `ASYM_GetFileDate()` function is similar to `getFileDate32()`, but it handles localization and dates beyond the year `2000` more efficiently, because you can specify the format of the return value.

PARAMETERS

PARAMETER	DESCRIPTION
`<path>`	The path and file name for which you want to set the time and date.

EXAMPLES

```
x = getFileDate32("c:\boot.txt")
```

getFileList()

TBDOS.DLL File Function (List)

SYNTAX `getFileList(<file name>)`

DESCRIPTION Returns a list of files matching a path and file name specification.

The function will return short file names only. Use `getFileListLFN()` if you need to return long file names.

RETURNS If no error occurs and if ToolBook finds files that match `<file name>`, `getFileList()` returns a list of matching files separated by `CRLFs`.

Otherwise, the function returns `null` and `sysError` is set to one of these values:

```
 -2  File name was invalid.
 -3  Path was not found.
-18  Matching file.
-20  Memory allocation error.
```

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    STRING getFileList(STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getFileList32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	The name of the file to find. The name can contain the wildcard characters `?` and `*`.

EXAMPLES

```
allFiles = getFileList("c:\project\*.*")
```

getFileList32()

TBFILE32.DLL File Function (List)

SYNTAX `getFileList32(<file name>)`

DESCRIPTION Returns a list of files matching a path and file name specification.

RETURNS If no error occurs and if ToolBook finds files that match `<file name>`, `getFileList()` returns a list of matching files separated by `CRLFs`.

Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 `getFileAttributes32()` is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    STRING getFileList32(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	The name of the file to find. The name can contain the wildcard characters `?` and `*`.

EXAMPLES

```
allFiles = getFileList32("c:\project\*.*")
```

getFileListLFN()

TBDOS.DLL File Function (List)

SYNTAX `getFileListLFN(<file name>)`

DESCRIPTION Returns a list of files matching a path and file name specification.

RETURNS If no error occurs and if ToolBook finds files that match `<file name>`, `getFileList()` returns a list of matching files separated by `CRLFs`.

Otherwise, the function returns `null` and `sysError` is set to one of these values:

`-2` File name was invalid.
`-3` Path was not found.
`-18` Matching file.
`-20` Memory allocation error.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    STRING getFileListLFN(STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getFileList32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	The name of the file to find. The name can contain the wildcard characters ? and *.

EXAMPLES

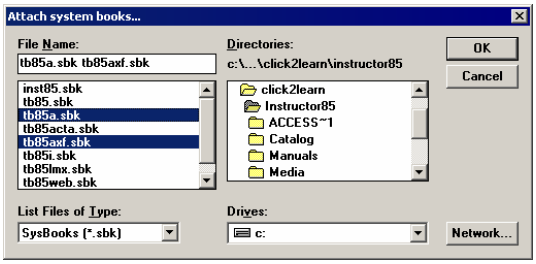
```
allFiles = getFileListLFN("c:\project\*.*")
```

getFileListDlg()

TBDLG.DLL File Function (Dialog)

SYNTAX `getFileListDlg(<caption text>,<default file>,<default path>,<filters>, <index>)`

DESCRIPTION Displays the standard Windows File List dialog box (similar to the Open File dialog box) that provides multi-select behavior.



RETURNS If no error occurs, the function returns a comma-separated list of items.

The first item is the current directory when the dialog box was closed, and is followed by a list of the selected file names.

The file name items may contain a path relative to the path of the first item, for example:

`c:\windows,win.ini,system.ini,..\tb\INSTRUCTOR.INI`

Otherwise, the function returns `null` and sets `sysError` to one of these values:

`-1` User canceled the dialog box.
`-2` Memory allocation error.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING getFileListDlg(STRING,STRING,STRING,STRING,INT)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `openFileDlg32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<caption text>`	The text displayed in the title bar of the dialog box.
`<default file>`	The default file name. If `null`, all files matching the specified filter are shown.
`<default path>`	The default path. If `null`, the current directory is used.
`<filters>`	A list of file name filters. Each filter is a two-item list containing a description and a wildcard, for example "`Icon (*.ico), *.ico`". This parameter can contain multiple pairs of filters.
`<index>`	An index into the `<filters>` list specifying which to use as the default. The first filter is `1`. To get the filter value, use the `getFileListDlgFilterIndex()` function.

EXAMPLES

```
to handle buttonClick
    filterList = "SysBooks (*.sbk),*.sbk"
    fileList = getFileListDlg("Attach system books...",null,null,
      filterList,1)
    if fileList is null
        request "User Canceled."
    else
        requestString = "Files selected: "
        pop fileList into path
        while itemCount(fileList) > 0
            pop fileList
            put path & "\" & IT & "," after requestString
        end
        clear last char of requestString
        request requestString
    end
end
```

getFileListDlgFilterIndex()

TBDLG.DLL File Function (Dialog)

SYNTAX `getFileListDlgFilterIndex(<window handle>)`

DESCRIPTION Returns the index number of the filter selected in the Windows File List dialog box the last time it was called in the current instance of ToolBook.

RETURNS If no error occurs, `getFileListDlgFilterIndex()` returns the index number of the specified dialog box's filter.

Otherwise, the function returns this value:

`-1` Requested index filter was not set for the current ToolBook instance.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    INT getFileListDlgFilterIndex(WORD)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<window handle>	A window handle of a viewer in the current instance of ToolBook.

EXAMPLES

```
to handle buttonClick
    get getFileListDlg("test","",".", \
        "ToolBook (*.tbk), *.tbk, All files (*.*), *.*", 1)
    vFilterNum = getFileListDlgFilterIndex(windowHandle of this window)
    request vFilterNum
end
```

getFileOnlyList()

TBDOS.DLL File Function (List)

SYNTAX

```
getFileOnlyList(<path>,<file attributes>,<sort order>)
```

DESCRIPTION

Returns a list of matching files based on the specified path, file name, and file attributes.

This function will return short file names only. Use `getFileOnlyListLFN()` if you need to return long file names.

RETURNS

If no error occurs, `getFileOnlyList()` returns a `CRLF`-separated list of files. One file appears on each line.

If an error occurs, the function returns `null` and sysError is set to one of these values:

```
 -2  Specified path was invalid.
 -3  Specified file attribute was invalid.
 -4  Specified sort order was invalid.
-20  Memory allocation error.
```

NOTES

You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    STRING getFileOnlyList(STRING,STRING,STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getFileOnlyList32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
<path>	A string that contains the name of the path and file you want. The string can contain `?` and `*` wildcards.
<file attributes>	A string containing a character that specifies the attributes of the specified file. You can use one of the following values: `D` = directory `NULL or ""` = defaults `H` = hidden `RA` = read-only and archive `S` = system
<sort order>	A string containing a character that specifies the sort order for the returned list. The values for `<sort order>` use the same characters as the DOS `DIR` command. You can use one of the following values: `N` = by name `D` = by date `E` = by extension `NULL or ""` = unsorted `S` = by size If the `<sort order>` parameter is preceded by a hyphen (`-`), the sort order is reversed.

EXAMPLES

```
fList = getFileOnlyList("c:\*.*", "",N)
```

getFileOnlyList32()

TBFILE32.DLL File Function (List)

SYNTAX `getFileOnlyList32(<path>,<file attributes>,<sort order>)`

DESCRIPTION Returns a list of matching files based on the specified path, file name, and file attributes.

RETURNS If no error occurs, `getFileOnlyList32()` returns a `CRLF`-separated list of files. One file appears on each line.

If an error occurs, the function returns `null` and sysError is set to one of these values:

Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 `getFileOnlyList32()` is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    STRING getFileOnlyList32(STRING,STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<path>`	A string that contains the name of the path and file you want. The string can contain `?` and `*` wildcards.
`<file attributes>`	A string containing a character that specifies the attributes of the specified file. You can use one of the following values: `D` = directory `H` = hidden `S` = system `NULL or ""` = defaults `RA` = read-only and archive
`<sort order>`	A string containing a character that specifies the sort order for the returned list. The values for `<sort order>` use the same characters as the DOS `DIR` command. You can use one of the following values: `N` = by name `E` = by extension `S` = by size `D` = by date `NULL or ""` = unsorted If the `<sort order>` parameter is preceded by a hyphen (`-`), the sort order is reversed.

EXAMPLES

```
fList = getFileOnlyList32("c:\*.*", "",N)
```

getFileOnlyListLFN()

TBDOS.DLL File Function (List)

SYNTAX `getFileOnlyListLFN(<path>,<file attributes>,<sort order>)`

DESCRIPTION Returns a list of matching files based on the specified path, file name, and file attributes.

RETURNS If no error occurs, `getFileOnlyListLFN()` returns a `CRLF`-separated list of files. One file appears on each line.

If an error occurs, the function returns `null` and sysError is set to one of these values:

- `-2` Specified path was invalid.
- `-3` Specified file attribute was invalid.
- `-4` Specified sort order was invalid.
- `-20` Memory allocation error.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    STRING getFileOnlyListLFN(STRING,STRING,STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getFileOnlyList32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<path>`	A string that contains the name of the path and file you want. The string can contain `?` and `*` wildcards.
`<file attributes>`	A string containing a character that specifies the attributes of the specified file. You can use one of the following values: `D` = directory `H` = hidden `S` = system `NULL or ""` = defaults `RA` = read-only and archive
`<sort order>`	A string containing a character that specifies the sort order for the returned list. The values for `<sort order>` use the same characters as the DOS `DIR` command. You can use one of the following values: `N` = by name `E` = by extension `S` = by size `D` = by date `NULL or ""` = unsorted If the `<sort order>` parameter is preceded by a hyphen (`-`), the sort order is reversed.

EXAMPLES

```
fList = getFileOnlyListLFN("c:\*.*", "",N)
```

getFileSize()

TBDOS.DLL File Function (Attribute)

SYNTAX

```
getFileSize(<file name>)
```

DESCRIPTION Gets the size of the specified file.

RETURNS If no error occurs, returns the file's size in `bytes`.

Otherwise, the function returns:

`-2` Specified file was not found.
`-3` Specified path was invalid.
`-20` Memory allocation error.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    LONG getFileSize(STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getFileSize32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	The name of the file with the attributes you want.

EXAMPLES

```
x = getFileSize(fName)
```

getFileSize32()

TBFILE32.DLL File Function (Attribute)

SYNTAX `getFileSize32(<file name>)`

DESCRIPTION Gets the size of the specified file.

RETURNS If no error occurs, returns the file's size in `bytes`.

Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    LONG getFileSize32(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	The name of the file with the attributes you want.

EXAMPLES

```
x = getFileSize32(fName)
```

getFileVersion()

TBDOS.DLL File Function (Attribute)

SYNTAX `getFileVersion(<file name>)`

DESCRIPTION Returns information from the `VERSIONINFO` resource of a specified file.

RETURNS If the function is successful, it returns a `CRLF`-delimited list that contains the following items:

```
Path where the file was found
File version
Language
Character set
Internal name
Product name
Product version
Original file name
Legal copyright
Legal trademark
```

A blank textline indicates `null` or a missing item.

If the function is not successful, it returns `null` and sets `sysError` to one of these values:

`-2`	Specified file was not found.	`-20`	Memory allocation error.
`-3`	Specified path was invalid.	`-30`	No VERSIONINFO resource was available

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    STRING getFileVersion(STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getFileVersion32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	The name of the file with the attributes you want.

EXAMPLES

```
x = getFileVersion(name of this book)
```

getFileVersion32()

TBFILE32.DLL File Function (Attribute)

SYNTAX `getFileVersion32(<file name>)`

DESCRIPTION Returns information from the `VERSIONINFO` resource of a specified file.

RETURNS If the function is successful, it returns a `CRLF`-delimited list that contains the following items:

```
Path where the file was found
File version
Language
Character set
Internal name (such as the module name for a DLL)
Product name
Product version
Original file name
Legal copyright
Legal trademark
```

A blank textline indicates `null` or a missing item.

If the function is not successful, it returns `null` and sets `sysError` to one of these values:

`-8` File not found.
`-27` Out of memory.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    STRING getFileVersion32(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	The name of the file with the attributes you want.

EXAMPLES

```
x = getFileSize32(fName)
```

getFreeDiskSpace()

TBDOS.DLL File Function (General)

SYNTAX `getFreeDiskSpace(<disk drive>)`

DESCRIPTION Gets the number of free bytes on the specified disk drive.

RETURNS If no error occurs, `getFreeDiskSpace()` returns the number of free bytes on the specified disk.

Otherwise, the function returns a negative number.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    DWORD getFreeDiskSpace(STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getFreeDiskSpace32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<disk drive>`	The letter designating the disk drive for which you want the number of free bytes.

EXAMPLES

```
freeSp = getFreeDiskSpace("c")
```

getFreeDiskSpace32()

TBFILE32.DLL File Function (General)

SYNTAX `getFreeDiskSpace(<disk drive>)`

DESCRIPTION Gets the number of free bytes on the specified disk drive.

RETURNS If no error occurs, `getFreeDiskSpace32()` returns the number of free bytes on the specified disk.

Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    DWORD getFreeDiskSpace32(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<disk drive>`	The letter designating the disk drive for which you want the number of free bytes.

EXAMPLES

```
freeSp = getFreeDiskSpace32("c")
```

getIniVar()

TBWIN.DLL INI Functions

SYNTAX `getIniVar(<section name>,<item name>,<file name>)`

DESCRIPTION Returns the value of the specified item in the specified section of any `.INI` file.

RETURNS If no error occurs, this function returns the value of the specified item in the specified section in the `.INI` file.

Otherwise, the function returns:

`-20` Memory allocation error.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    STRING getIniVar(STRING,STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<section name>`	The name of the section in the .INI file that contains the item with the value you want.
`<item name>`	The name of the item with the value you want.
`<file name>`	The name of the .INI file.

EXAMPLES

```
val = getIniVar("General","lastScore","student005.ini")
```

getLongFileName32()

TBFILE32.DLL File Function (Filename)

SYNTAX `getLongFileName32(<file name>,<full path>)`

DESCRIPTION Returns a long filename specification of a file reference.

RETURNS This function will only return the long file specification for an existing file. If `<file name>` does not exist, the function will return `NULL`.

If no error occurs and if ToolBook finds a long file specification that matches `<file name>`, `getLongFileName32()` returns a long file specification.

Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    STRING getLongFileName32(STRING, WORD)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	The filename and path.
`<full path>`	Specifies the part of the file specification to convert to a long filespec. When `<full path>` = `1`, the entire filespec will be converted to a long filespec. When `<full path>` = `0`, only the last entry in the filespec will be converted and returned.

EXAMPLES

```
x = "c:\project\logboo~1.txt"
request getLongFileName32(x,1)
```

getObjectList()

Object Function

SYNTAX `getObjectList(<object scope>,<object types>,<property filter>)`

DESCRIPTION Returns a list of object references for the book within the specified object scope, of the specified object type(s), and having the specified property.

PARAMETERS

PARAMETER	DESCRIPTION
`<object scope>`	A valid reference to a ToolBook object (`background`, `page`, `group`, `field`, or `recordfield`) that can contain other objects. Cannot be the `book`.
`<object types>`	A string that contains a list of object types, for example: `"button,comboBox,hotword"`. To automatically include all types, you can specify a null string (`""`).

PARAMETERS

PARAMETER	DESCRIPTION
`<property filter>`	A string containing the name of a property by which to filter the search. If not `NULL`, an object must have a non-`NULL` value in the property by this name in order to be included in the returned list of objects. If `NULL`, any object of the appropriate type and in the appropriate scope is included in the list. **NOTE**: For backward compatibility, `true` and `false` are also valid values for this parameter. If `true`, an object is disregarded if it does not contain a script. If `false`, an object is included regardless of whether it contains a script. Note that passing `true` is equivalent to passing `"script"`, and passing `false` is equivalent to passing `NULL`.

EXAMPLES

```
-- Returns all buttons on this page that have a script
getObjectList(this page,"button",true)

-- Returns all objects on this page that have a tool tip defined
getObjectList(this page,"","ASYM_ToolTip")

-- Returns all objects on this page that have actions defined for
-- the buttonClick event
getObjectList(this page,"","ASYM_Actions(buttonClick)")
```

getOpenFileDlgFilterIndex()

TBDLG.DLL File Function (Dialog)

SYNTAX

```
getOpenFileDlgFilterIndex(<window handle>)
```

DESCRIPTION

Returns the index number of the filter selected in the Open File dialog box the last time it was called in the current instance of ToolBook.

RETURNS

If no error occurs, `getFileListDlgFilterIndex()` returns the index number of the specified dialog box's filter.

Otherwise, the function returns this value:

`-1` Requested index filter was not set for the current ToolBook instance.

NOTES

You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    INT getOpenFileDlgFilterIndex(WORD)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getOpenFileDlgFilterIndex32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<window handle>`	A window handle of a viewer in the current instance of ToolBook.

EXAMPLES

```
to handle buttonClick
    get OpenFileDlg("test","",".", \
      "ToolBook (*.tbk),*.tbk,All files (*.*),*.*",1)
    vFilterNum = getOpenFileDlgFilterIndex(windowHandle of this window)
    request vFilterNum
end
```

getOpenFileDlgFilterIndex32()

TBFILE32.DLL File Function (Dialog)

SYNTAX `getOpenFileDlgFilterIndex32(<window handle>)`

DESCRIPTION Returns the index number of the filter selected in the Open File dialog box the last time it was called in the current instance of ToolBook.

RETURNS If no error occurs, `getFileListDlgFilterIndex32()` returns the index number of the specified dialog box's filter.

Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    INT getOpenFileDlgFilterIndex32(WORD)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<window handle>`	A window handle of a viewer in the current instance of ToolBook.

EXAMPLES

```
to handle buttonClick
    get OpenFileDlg("test","",".", \
      "ToolBook (*.tbk),*.tbk,All files (*.*),*.*",1)
    vFilterNum = getOpenFileDlgFilterIndex(windowHandle of this window)
    request vFilterNum
end
```

getParameter()

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided function that returns a named parameter from a URL or ToolBook command line.

The parameters should be in the form `<name>=<value>` pairs separated by spaces. The names and values cannot contain spaces but standard URL character replacements are supported.

NOTES This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

EXAMPLES

```
On click...
  Set text of Field "info" to getParameter("uName")
```

getProperty()

Object Function

SYNTAX `getProperty(<objRef>,<propertyName>)`

DESCRIPTION Returns the current value stored in an object's property.

RETURNS The value stored in the named property.

If `propertyName` is invalid or `NULL`, the function will return `NULL`.

If `<objRef>` is invalid, the function will return `NULL`, and will set `sysError` to an appropriate message.

NOTES When using `getProperty()` or `setProperty()`, the object's property will be accessed directly.

You cannot use the `to get` or `to set` handler structure to intercept these functions in OpenScript.

PARAMETERS

PARAMETER	DESCRIPTION
<objRef>	A reference to the object from which the property value will be retrieved.
<propertyName>	A string representing the name of the property to return.

EXAMPLES

```
-- This handler populates a "Property Report" field with a list of
-- all of the properties and property values for an object
to handle propertyReport pObj
    refReportField = field "Property Report"
    clear text of refReportField
    propList = propertyList(pObj)
    while propList <> NULL
        pop propList into propName
        propValue = getProperty(pObj,propName)
        put propName & TAB & propValue & CRLF after text of refReportField
    end
end
```

getPropertyDimensions()

Object Function

SYNTAX `getPropertyDimensions(<objRef>,<property Name>)`

DESCRIPTION Returns a comma-separated list of dimensions for the array stored in the object's property or user property. `NULL` if the property does not exist or if the property is not an array.

PARAMETERS

PARAMETER	DESCRIPTION
<objRef>	A reference to the object to retrieve the property value from.
<property Name>	A string representing the name of the property or user property.

EXAMPLES

```
-- Returns NULL if the button does NOT have a hyperlink
get getPropertyDimensions(button "Next", "_asym_hyperlinks")

-- This function is most useful for determining if a property contains an
-- array before attempting to use it. For example, if an object has a user
-- property that may contain an array, or could be NULL, you have to find out
-- before you can assign the property's value to a variable:

-- If a button has a property named nav that DOES contain a one-dimensional
array value
local a
local b[]
-- This will succeed
b = nav of button "Previous"
-- This will generate an Execution Suspended Message
a = nav of button "Previous"

-- If a button has a property nav that is NULL, or does not contain an array
local a
local b[]
-- This will succeed
a = nav of button "Previous"
-- This will generate an Execution Suspended Message
b = nav of button "Previous"

-- In both cases, an ES can be avoided by checking the dimensions first:
local a
local b[]
if getPropertyDimensions(button "Previous", "nav") <> NULL
    b = nav of button "Previous"
else
    a = nav of button "Previous"
end
```

getSaveAsDlgFilterIndex()

TBDLG.DLL File Function (Dialog)

SYNTAX `getSaveAsDlgFilterIndex(<window handle>)`

DESCRIPTION Returns the index number of the filter selected in the Save File dialog box the last time it was called in the current instance of ToolBook.

RETURNS If no error occurs, `getSaveAsDlgFilterIndex()` returns the index number of the specified dialog box's filter.

Otherwise, the function returns this value:

`-1` Requested index filter was not set for the current ToolBook instance.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    INT getSaveAsDlgFilterIndex(WORD)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `getSaveAsDlgFilterIndex32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<window handle>`	A window handle of a viewer in the current instance of ToolBook.

EXAMPLES

```
to handle buttonClick
    get saveAsDlg("test","",".", \
      "ToolBook (*.tbk),*.tbk,All files (*.*),*.*",1)
    vFilterNum = getSaveAsDlgFilterIndex(windowHandle of this window)
    request vFilterNum
end
```

getSaveAsDlgFilterIndex32()

TBFILE32.DLL File Function (Dialog)

SYNTAX `getSaveAsDlgFilterIndex32(<window handle>)`

DESCRIPTION Returns the index number of the filter selected in the Save File dialog box the last time it was called in the current instance of ToolBook.

RETURNS If no error occurs, `getSaveAsDlgFilterIndex()` returns the index number of the specified dialog box's filter. Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    INT getSaveAsDlgFilterIndex32(WORD)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<window handle>`	A window handle of a viewer in the current instance of ToolBook.

EXAMPLES

```
to handle buttonClick
    get saveAsDlg("test","",".", \
      "ToolBook (*.tbk),*.tbk,All files (*.*),*.*",1)
    vFilter = getSaveAsDlgFilterIndex32(windowHandle of this window)
    request vFilter
end
```

getShortFileName32()

TBFILE32.DLL File Function (Filename)

SYNTAX `getShortFileName32(<file name>,<full path>)`

DESCRIPTION Returns a short filename specification of a file reference.

The short file name is always `12` characters or fewer, but the full file specification can be up to `256` characters in length.

RETURNS This function will only return the short file specification for an existing file. If `<file name>` does not exist, the function will return `NULL`.

If no error occurs and if ToolBook finds a short file specification that matches `<file name>`, `getShortFileName32()` returns a short file specification.

Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    STRING getShortFileName32(STRING,WORD)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	The filename and path.
`<full path>`	Specifies the part of the file specification to convert to a short filespec. When `<full path>` = `1`, the entire filespec will be converted to a short filespec. When `<full path>` = `0`, only the last entry in the filespec will be converted and returned.

EXAMPLES

```
x = name of this book
request getshortFileName32(x,1)
```

getWinIniVar()

TBWIN.DLL INI Functions

SYNTAX `getWinIniVar(<section name>,<item name>)`

DESCRIPTION Returns the value of the specified item in the specified section of the `WIN.INI` file.

RETURNS If no error occurs, this function returns the value of the specified item in the specified section in the `WIN.INI` file.

Otherwise, the function returns:

`-20` Memory allocation error.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    STRING getWinIniVar(STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<section name>	The name of the section in the WIN.INI file that contains the item with the value you want.
<item name>	The name of the item with the value you want.

EXAMPLES

```
setting = getWinIniVar("desktop","WallPaper")
```

go

Navigation

SYNTAX

```
go [to] <expression>
```

DESCRIPTION

Goes to any page in the currently open book, or to any page in any other book.

If the destination is a book, and a page is not specified, ToolBook goes to the first page of the book.

NOTES

If the `go next page` statement is encountered on the last page of a book, ToolBook goes to the first page.

If the `go previous page` statement is encountered on the first page of a book, ToolBook goes to the last page.

PARAMETERS

PARAMETER	DESCRIPTION
<expression>	A valid identifier for a page or book, including relational values (such as first or last) to identify a page.

EXAMPLES

```
go to book "c:\mybooks\contacts.tbk"
```

```
go first page of this background
```

graphic

String Operators

SYNTAX

```
graphic of <expression> = <resourceRef>
```

DESCRIPTION

A string operator used to assign a graphic resource as a single character in text field. Also to return a reference to the graphic resource assigned to a single character in a text field.

Use `graphic` at the beginning of an expression that refers to a character in a text field.

NOTES

When you set the `graphic` of a character in a field, the graphic takes the place of the character. The character itself will not be visible unless you clear the `graphic` of the character.

PARAMETERS

PARAMETER	DESCRIPTION
<expression>	An expression that results in a single character reference to text in a text field.
<resourceRef>	An expression that results in a reference to a ToolBook graphical resource.

EXAMPLES

```
graphic of char 1 of text of field ID 0 = icon "red bullet"
```

```
graphic of char 1 of text of hotword "SoundClip" = bitmap "speaker"
```

```
res = graphic of char 5 of text of field "images"
if res = bitmap "pencil"
    graphic of char 5 of text of field "images" = bitmap "eraser"
end
```

gray

Color Constants

DESCRIPTION Represents the color `gray`. This is equivalent to the `RGB` value of `128,128,128` and the `HLS` value of `0,50,0`.

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES

```
rgbFill of field "list" = gray
```

```
if rgbStroke of rectangle "drop area" = gray
    rgbStroke of rectangle "drop area" = 0,191,0
end
```

```
strokeColor of ellipse id 14 = gray
```

green

Color Constants

DESCRIPTION Represents the color green. This is equivalent to the `RGB` value of `0,255,0` and the `HLS` value of `120,50,100`.

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES

```
rgbFill of field "list" = green
```

```
if rgbStroke of rectangle "drop area" = green
    rgbStroke of rectangle "drop area" = 0,191,0
end
```

```
strokeColor of ellipse id 14 = green
```

group

Grouping Message

DESCRIPTION Sent to the page when you choose `Group` from the `Object` menu.

You can also `send` this message using the `send group` statement. ToolBook's default response is to make the selected objects into a `group` or, if there is no selection or only one selected object, to do nothing.

Once the group has been created, the value of `selection` becomes the group.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
draw field from 300,3000 to 2800,3600
extend select button "DistributePayments", rectangle "frame"
send group
```

group

Object Type

DESCRIPTION The object type name for a `group`, which is a collection of objects that has properties like a single object.

NOTES The parent of a group can be the page, background, or group. A group is the parent of any objects in the group, including other groups.

To create a group using OpenScript, select the objects you want to group, then use the `send group` statement.

TO REFER TO A GROUP	USE THIS SYNTAX
On the current page	`group "Controls"`
On the current background	`group ID 276 of this background`
On another page	`group "Shine" of page 23`
On another background	`group "radio7" of background "Seven"`
In another book	`group "Checklist" of page "Index" \` `  of book "order.tbk"`

The `destroy`, `make`, `moved`, and `sized` notification messages are sent automatically to groups. Only the group receives the notification message, not the objects in the group.

For mouse event messages, the message is first sent to the object, then forwarded to the group if there is no handler for the message in that object's script.

When a group is ungrouped, its `script`, `user properties`, and `name` are discarded.

hasPropertyDialog

Property

DESCRIPTION Determines if a control has its own built in property dialog.

NOTES This is useful if you are creating a widgetized object where a property sheet would be needed to permit users to configure the object. Many controls have their own built in property sheets and ToolBook is more than happy to display the controls own property sheet.

VALUES Returns `true` if the control has a custom property dialog, otherwise returns `false`.

EXAMPLES

```
get hasPropertyDialog of tList "mainList"
```

HDMediaPath

Property

DESCRIPTION A book property that specifies the hard disk directories in which ToolBook searches for external media referenced by `clips`.

NOTES You can get or set this property.

ToolBook uses the value of this property when a clip's `mmSearchHD` property is `true`.

Use this property to avoid path dependencies in your scripts and simplify the organization and delivery of clips in your application.

VALUES A comma separated list of directories.

Add the string value `<BookPath>` to indicate the current book's path.

For best performance, restrict the hard disk path to only one directory.

EXAMPLES

```
driveLetter = fetchTheDriveLetter()
baseCDPath = ":\allmedia\"
HDMediaPath = driveLetter & baseCDPath
```

header

Property

DESCRIPTION A book property that specifies the contents of a header to be printed on each sheet of a report.

NOTES You can get or set this property.

VALUES A string of up to 32,000 characters.

The default is `null` (no header).

To specify a `print time`, `print date`, or `running page number`, enter `~t`, `~d`, or `~p`, respectively.

The font of the footer is determined by the values for `sysFontFace`, `sysFontSize`, and `sysFontStyle` when the footer is set.

EXAMPLES

```
header of this book = "Draft Date: ~d      Page: ~p"
```

height

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of almost all ToolBook object types that specifies the current height of that object.

NOTES You can get or set this property in OpenScript.

You can get but not set the property in the Actions Editor.

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

VALUES A whole numbers that specifies the `height` in `page units`.

EXAMPLES

```
height of button "foo" = 500
```

```
x = height of button "foo"
```

hidden

Notification Message

DESCRIPTION Sent to a `viewer` when it is hidden.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle hidden
    focusWindow = mainWindow
    forward
end
```

hiddenByHideOnReader

UNDOCUMENTED Property

DESCRIPTION A viewer property that specifies if a viewer is currently hidden as a result of toggling to `Reader` level.

NOTES You can get or set this property.

If you set the `hideOnReader` property of a viewer to `true`, and then toggle to `Reader` level, the viewer will hide itself.

VALUES `True` or `false`.

If the viewer was hidden from the act of toggling to `Reader` level, the `hiddenByHideOnReader` property will be set to `true`.

If the viewer was hidden for other reasons, such as setting the `visible` property of the viewer to `false`, the `hiddenByHideOnReader` property will be set to `false`.

EXAMPLES

```
x = hiddenByHideOnReader of viewer "help"
```

hide

Viewer & Object Command

SYNTAX

```
hide [<object>]
```

DESCRIPTION Removes an object from view. The result of this command is the same as setting the `visible` property for the object to `false`.

For viewer objects, the hide command does not close the viewer.

NOTES You cannot `select` a hidden object.

PARAMETERS

PARAMETER	DESCRIPTION
`<object>`	A reference to any object or built-in ToolBook system object, such as a palette, that has a `visible` property. If no `<object>` is indicated, the current selection is `hidden`.

EXAMPLES

```
hide recordfield "Security level" of this background
```

```
hide commandWindow
```

```
hide viewer "Tools"
```

hideOnDeactivate

Property

DESCRIPTION A property of a viewer that specifies whether a viewer is hidden when its application is `deactivated`.

An application is `deactivated` when another application becomes active or the user clicks the desktop.

NOTES You can get or set this property.

A viewer with `hideOnDeactivate` set to `true` is only visible while its ToolBook application is active.

The `hideOnDeactivate` property is especially useful for creating a viewer that is used as a `palette` because the viewer can automatically be hidden while a window from another application as the focus.

Set `hideOnDeactivate` to `false` to create a viewer that remains available while you are working in different books or applications.

VALUES `True` or `false`.

The default is `true`.

If `true`, the viewer is hidden while its application is inactive.

If `false`, the viewer remains visible even when its application is no longer active. ToolBook automatically shows the viewer when its application becomes active again.

EXAMPLES

```
hideOnDeactivate of viewer "help" = true
```

hideOnReader

Property

DESCRIPTION A viewer property that specifies whether a viewer is hidden on the transition from `Author` level to `Reader` level.

NOTES You can use get or set with this property.

When this property is `true`, a viewer visible at `Author` level will be hidden on the transition to `Reader` level.

A viewer that was hidden in the `Author` to `Reader` transition will be shown again on the `Reader` to `Author` transition.

VALUES `True` or `false`.

The default is `false`.

EXAMPLES

```
hideOnReader of viewer "help" = true
```

highlight

Property

DESCRIPTION A button property that specifies whether the button `inverts` when clicked.

A hotword property that specifies whether a hotword `flashes` briefly when it is clicked.

NOTES You can get or set this property.

VALUES `True` or `false`. The default is `true` for buttons and `false` for hotwords.

EXAMPLES

```
highlight of button "tar3314" = false
highlight of hotword "pens" of field "x3" = true
```

HLStoRGB()

TBWIN.DLL Color Functions

SYNTAX

```
HLStoRGB(<hue>,<lightness>,<saturation>)
```

DESCRIPTION Converts the specified `HLS` color value into an `RGB` color value.

RETURNS If no error occurs, `HLStoRGB()` returns a string of three values that represent the `red`, `green`, and `blue` values. Otherwise, the function returns:

`-20` Memory allocation error.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING HLStoRGB(DOUBLE,DOUBLE,DOUBLE)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<hue>`	A hue value in the range 0 to 360.
`<lightness>`	A lightness value in the range 0 to 100.
`<saturation>`	A saturation value in the range 0 to 100.

If any parameter is out of range, the function assumes a value of `0` for that parameter.

EXAMPLES

```
clr = HLStoRGB(50,20,50)
```

horizontalDisplayRes()

TBWIN.DLL Screen Display Function

SYNTAX `horizontalDisplayRes()`

DESCRIPTION Returns the horizontal resolution of the display device in `pixels` (Screen Width).

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT horizontalDisplayRes()
end
```

PARAMETERS No parameters

EXAMPLES

```
-- returns 800 on a 800x600 display
val = horizontalDisplayRes()
```

horizontalDisplaySize()

TBWIN.DLL Screen Display Function

SYNTAX `horizontalDisplaySize()`

DESCRIPTION Returns the horizontal resolution of the display device in `millimeters` (Screen Width).

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT horizontalDisplaySize()
end
```

PARAMETERS No parameters

EXAMPLES

```
val = horizontalDisplaySize()
```

hotword

Object Type

DESCRIPTION The object type name for a `hotword`, which is text that has an `ID`, a script, and properties like other objects.

NOTES Hotwords in recordfields are stored on the page, like other recordfield text.

The parent of the hotword is the field or recordfield that it appears in.

To create a hotword using OpenScript, select the text in a field or recordfield for the hotword, then use the `send createHotword` statement.

TO REFER TO A HOTWORD	USE THIS SYNTAX
On the current page	`hotword "Local Names"` `hotword ID 276`
On another page	`hotword "Shine" of page 23`
On another background	`hotword "Yes" of background "Seven"`
In another book	`hotword "Item" of page "Index" of \` `  book "order.tbk"`

To find a hotword's `position` in a field, get the value of the hotword's `textOffset` property.

To see a list of currently selected hotwords in the book, use the `selectedHotwords` system property.

To see a list of the hotwords in a field, get the value of the field's `objects` property. The hotwords are listed in the order they appear in the field. To identify a newly created hotword, find the item with the highest `ID` number in the list.

To show the hotword style for hotwords in all fields in a book, send the `showHotwords` message. The hotwords are shown in the current `hotwordStyle` setting for the book. The default `hotwordStyle` for books is `color`.

hotwordColor

Property

DESCRIPTION A book property that specifies the default color for hotwords.

NOTES You can get or set this property.

When `sysHotwordsShown` is `true`, hotwords are shown in the current `hotwordStyle` setting for the book.

ToolBook displays hotwords in the default hotword color of the book when `sysHotwordsShown` is `true` and `hotwordStyle` is `color` (the default setting for books).

To create different color hotwords, set the `hotwordStyle` of the book to `none`, and set the `strokeColor` of each individual hotword.

VALUES An `HLS` color represented as a list of three non-negative numbers, or a valid reference to a ToolBook color constant.

The default value is 0,50,100 (or `red`).

EXAMPLES

```
hotwordColor of this book = green
```

hotwordStyle

Property

DESCRIPTION A book property that specifies the style for all hotwords within a book.

A hotword property that specifies the style for individual hotwords.

NOTES You can get or set this property.

A hotword's `hotwordStyle` property always takes precedence over the book's `hotwordStyle` property, unless the hotword's `hotwordStyle` has a value of `bookDefault`. In that case, it assumes the book's value for `hotwordStyle`.

VALUES FOR BOOKS: `color`, `frame`, or `none`; the default is `color`.

- If a book's `hotwordStyle` is `color`, the hotword text is drawn in the color specified by the book's `hotwordColor` property.
- If the value is `frame`, the hotword text is enclosed in a `rectangle`.
- If `none`, the hotword text matches the text of its field or recordfield.

FOR HOTWORDS: `bookDefault`, `color`, `dotted`, `frame`, `underline`, or `none`.

- The default is `bookDefault`. If `bookDefault`, the hotword assumes the value of the book's `hotwordStyle`.
- If `color`, the hotword text is drawn in the color specified by the book's `hotwordColor` property.
- If `frame`, the hotword text is enclosed in a rectangle.
- If `none`, the hotword text matches the text of its field or recordfield.

To create different color hotwords, set the `hotwordStyle` of the book to `none`, and set the `strokeColor` of each individual hotword.

EXAMPLES

```
hotwordStyle of this book = "frame"

hotwordStyle of hotword "Book Link" = "color"
```

hypotenuse()

Trigonometric Values

SYNTAX

```
hypotenuse(<length>,<length>)
```

DESCRIPTION

Returns the length of the hypotenuse of a right triangle, given the length of the other two sides.

PARAMETERS

PARAMETER	DESCRIPTION
`<length>`	An expression that yields a number.

EXAMPLES

```
sideA = 55
sideB = 66
hyp = hypotenuse(sideA,sideB)
```

icon

Resource Type

DESCRIPTION

The resource type name for an `icon` resource.

Icon resources are referenced in OpenScript by `name` or by `ID` using the `icon` keyword. The `ID` is determined by the ToolBook system, but you can assign any `name` to the resource.

You can set an `icon` resource for the `icon` property of viewer objects; for the `checkedGraphic`, `disabledGraphic`, `invertGraphic`, and `normalGraphic` properties of a graphic button; or for the `dragImage` and `noDropImage` object properties.

EXAMPLES

```
icon of mainWindow = icon "project"

invertGraphic of button "car" = icon "door"
```

icon

Property

DESCRIPTION A property of a viewer that specifies the `icon` used when a viewer (including the Main window) is minimized.

NOTES You can get or set this property.

Setting the `icon` property of the book is the same as setting the `icon` property of the `mainWindow`.

If you save the book as an `.EXE` file, ToolBook adds the icon into the file header so that Windows can use it as your file icon.

If you set the icon property to a file name string that specifies a valid Windows icon, ToolBook imports the icon file as a resource and sets the icon property to that resource reference.

ToolBook is apparently limited to these icon types: `32 x 32` `pixel` icons at `16` or `256` colors, `16 x 16` `pixel` icons at `16` or `256` colors.

VALUES A valid file name or reference to an icon resource, or `null`.

If `null`, the default ToolBook icon is assigned to the viewer.

EXAMPLES

```
-- use an icon from another book
icon of viewer "folder" = icon "folder" of book "icons.tbk"

-- use an icon from a file
icon of viewer "folder" = "c:\media\folder.ico"
```

idle

Notification Message

DESCRIPTION Sent continuously when no other actions are occurring at `Reader` level.

The `idle` message is sent to the current page.

You can determine how often the idle message is sent by configuring the `sysIdleDelay` setting.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
-- Put this in page Script (or higher) to create a timer on the page
to handle idle
    get text of field "timer"
    if IT <> sysTime
        set text of field "timer" to sysTime
    end
    forward
end

-- Place this notify handler in a field to allow the field to
-- get the message and manage its own script
notifyAfter idle
    get my text
    if IT <> sysTime
        set my text to sysTime
    end
    forward
end
```

idNumber

Property

DESCRIPTION A property of all ToolBook object types that is used as an `identification number`.

NOTES You cannot set this property.

If you import a book or cut and paste objects or pages, the `idNumber` values for those objects will change.

Typically it is not a good idea to refer to an object by its `idNumber`. If you want to refer to an object, name the object and then refer to the object by name.

If by chance you create enough objects that the next `idNumber` to be created would be higher than the max value of `65535`, the numbering will start back at `0` again.

VALUES An whole number in the range `0` to `65535` that is unique to each page in a book, each background in a book, each object on a page or background, and each resource type in a book. ToolBook assigns this number when the object is created.

EXAMPLES

```
-- not recommended to refer to objects by idNumber
-- but here is a sample anyway
if idNumber of target = 14
    position of target = 0,0
end
```

if/then/else

Control Structure

SYNTAX

```
if <expression> [then]
    <statements>
[else
    <statements>]
end [if]
```

DESCRIPTION Allows statements to be executed depending on the value of `<expression>`.

NOTES If the `expression` evaluates to `true`, ToolBook executes the `then` statements.

If the `expression` evaluates to `false`, ToolBook executes the `else` statements if there is one, or skips to the `end if` statement if there is no `else` clause.

Do Not use the `break` command to attempt to break out of a `if/then/else` control structure as it will cause a break of the entire handler instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that evaluates to `true` or `false`.
`<statements>`	One or more OpenScript statements.

EXAMPLES

```
if x = 5 then
    request "correct"
else
    request "incorrect"
end if
```

```
if x = "Apple"
    request "correct"
end
```

imageBuffers

Property

DESCRIPTION A property of a viewer that specifies the number of image buffers that ToolBook allocates to a viewer when it displays a page.

NOTES You can get or set this property.

An image buffer is a memory bitmap into which ToolBook pre-draws a page's objects.

A setting of `0` or `1` provides acceptable results for viewers used as dialog boxes and palettes.

For best results when using animation on a page displayed in a viewer, set that viewer's `imageBuffers` property to `1`.

If you are moving, hiding, or showing objects on a page with a complex background, or creating navigation between pages with a complex background, set the viewer's `imageBuffers` property to `2`.

VALUES `0`, `1`, or `2`. The default is `2` for the `mainWindow` and `0` for all other viewers.

EXAMPLES

```
imageBuffers of viewer "help" = 1
```

imageInvalid

Property

DESCRIPTION A background or page property that indicates if an image has been changed since it was last stored.

When `imageInvalid` is `true`, a message appears in the Page Properties or Background Properties dialog box to indicate that the page or background has changed since its image was stored.

NOTES This is a read only property and cannot be set.

VALUE If `true`, the page or background has changed since the image was stored.

EXAMPLES

```
get imageInvalid of this page
```

import resource

Resource Commands

SYNTAX

```
import <type> resource <file name> [as <name>]
```

DESCRIPTION Imports a resource into a book.

The imported resource appears as an icon in the Resource Manager dialog box.

PARAMETERS

PARAMETER	DESCRIPTION
`<type>`	A valid name for a resource: `bitmap`, `cursor`, `font`, `icon`, `menuBar`, `palette`, or `sharedScript`. ToolBook also supports `clip`.
`<file name>`	A string that specifies the name of the file to import, including the path if necessary. ToolBook supports the following file formats: `.BMP`, `.DIB`, `.WMF`, `.CUR`, `.GIF`, `.ICO`, `JPG`, `.MNU`, `.PAL`, `.TTF`. or `.TXT`. ToolBook also supports `.CPF` for clip libraries. **NOTE** The `import resource` command only supports standard Windows bitmap formats (`BMP`, `DIB`, and `WMF`) at `runtime`, because other graphics filters are not redistributable.
`<name>`	A string assigned to the `name` property of the resource when it is imported. The resource can then be referenced by its `name` or `ID` number.

EXAMPLES

```
import icon resource "c:\plane.ico"
import cursor resource "tree.cur" as "tree"
import bitmap resource "c:\project\image.gif"
```

in

Control Structure

SYNTAX

```
in <viewerRef>
    <statements>
end [in]
```

DESCRIPTION

Executes statements within the context of the specified viewer. Use the `in` control structure to temporarily specify a different target window while a script is running.

NOTES

ToolBook changes the target window to the specified `<viewerRef>` upon entering the `in` control structure and executes each `statement` in the control structure within the context of that viewer.

When the script exits the in control structure, ToolBook reverts the value of `targetWindow` to its value before the in control structure was entered.

PARAMETERS

PARAMETER	DESCRIPTION
`<viewerRef>`	A reference to a viewer.
`<statements>`	One or more OpenScript statements.

EXAMPLES

```
-- Method without using the IN control structure
text of field "info" of currentPage of viewer "help" = null
fillColor of button "x3" of currentPage of viewer "help" = white
send buttonClick to button "x3" of currentPage of viewer "help"

-- Easier way, using the IN control structure
in viewer "help"
    text of field "info" = null
    fillColor of button "x3" = white
    send buttonClick to button "x3"
end
```

increment()

Arithmetic Values

SYNTAX

```
increment <expression> [by <amount>]
```

DESCRIPTION

Adds an amount to the value of an expression.

If the value of `<amount>` or value of `<expression>` is not a number ToolBook will generate an `Execution Suspend` error message.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	Any expression that yields a number.
`<amount>`	Any expression that yields a number. Default value is `1` if not specified.

EXAMPLES

```
to handle buttonClick
    score = 0
    step k from 1 to 10
        if fillColor of rectangle ("rect" & k) = yellow
            increment score
        end
    end
    request "Your total score is: " & score
end

-- Move this object to the right by 300 page units
increment item 1 of position of self by 300
```

indents

Property

DESCRIPTION A field or recordfield property that specifies the width of margin indents.

NOTES You can get or set this property.

VALUES A list of three non-negative numbers in `page units`, specifying the first line indent, the left margin indent, and the right margin indent.

The default is `0,0,0`.

EXAMPLES

```
-- Add a 3 pixel margin around text in field
indents of field "notepad" = 45,45,45

-- Create a hanging Outdent
indents of field "notepad" = 45,200,45
```

info_Description

User Property

DESCRIPTION A property that specifies the description of a selected object in the catalog.

NOTES When you add an object to the catalog, or modify an existing object, you can add or revise this text to describe the object.

VALUES A string of text.

info_Keywords

User Property

DESCRIPTION A list of keywords that describes the page when it is exported to HTML.

NOTES The keywords are placed in a META tag. They are also used during DHTML export to populate the `keyword` section of the manifest file.

VALUES Any string.

info_LastSaved

User Property

DESCRIPTION A book property that specifies the date and time when the book was last saved.

NOTES This value may be different from the file date because some utilities and copy functions change the file date.

VALUES A string that contains the date and time.

info_LastSavedBy

User Property

DESCRIPTION A book property that specifies the last user that saved the book.

NOTES By default, the authoring system book (TB89A.SBK) writes this information every time you save a file, using the `User Name` from the `[User Info]` section of the `ASYM.INI` file, if available.

VALUES A string that contains the name of the user.

info_Title

User Property

DESCRIPTION A book property that specifies an optional title for the book.

NOTES This property is typically used by dialog boxes that open templates, as well as Book Specialists that display the book title rather than a file name.

VALUES A string that specifies the title.

innerBevelWidth

Property

DESCRIPTION A stage property that specifies the width of the inner 3D bevel of its frame.

NOTES You can get or set this property.

VALUES A number in `page units`.

The default is `15`.

EXAMPLES

```
innerBevelWidth of stage "player" = 30
```

innerBounds

Property

DESCRIPTION A stage property that specifies the location of the stage's media display area in `page units`.

NOTES This is a read only property and cannot be set.

VALUES A list of four numbers in `page units`.

The first two numbers specify the `x,y` coordinates of the upper-left corner of the display area.

The last two numbers specify the `x,y` coordinates of the lower-right corner.

EXAMPLES

```
-- position a text field over player area of the stage
bounds of field "help" = innerBounds of stage "player"
```

into

Articles and Prepositions

DESCRIPTION Used with various OpenScript commands [such as `pop`, `push`, `insert graphic`] to indicate that a value should be placed into a container.

EXAMPLES

```
put "The End" into text of field "story"
```

```
insert graphic bitmap "Fog" into text of focus
```

```
pop mylist into goodList
```

invert

Property

DESCRIPTION A button property that specifies whether the button appears depressed.

A hotword property that specifies whether a hotword appears depressed.

NOTES You can get or set this property.

If invert is true, a button or hotword appears in its `highlighted` state.

VALUES `True` or `false`. The default is `false`

invertGraphic

Property

DESCRIPTION A button property that specifies a graphic resource assigned to be the resource shown when the button's `invert` property is set to `true` or the user clicks an enabled button and the `buttonDown` message is sent.

NOTES You can get or set this property.

VALUES A valid resource reference.

You can assign an `icon`, `cursor`, or `bitmap resource`.

If `null`, ToolBook displays the image that is set for the normal graphic.

EXAMPLES

```
invertGraphic of button "Print" = icon "PrinterChosen"
```

ipmt()

Financial Values

SYNTAX

```
ipmt(<rate>,<period>,<nper>,<present value>,<future value>[,<type>])
```

DESCRIPTION A financial function that returns the amount of interest to be paid on an investment or borrowed sum for a given period. This function is based on periodic, fixed payments and a constant interest rate. Payments out of pocket should be entered as negative values, while income should be entered as positive values.

PARAMETERS

PARAMETER	DESCRIPTION
`<rate>`	A positive whole number representing the interest rate per period.
`<period>`	A positive whole number representing the period for which the interest is to be calculated. This should be in the range `1` to `<nper>`.
`<nper>`	A positive whole number representing the total number of payment periods.
`<present value>`	A number representing the present value of the amount borrowed.
`<future value>`	A number representing the future value or a cash flow balance to be attained after the last payment is made. If `<future value>` is omitted, it is assumed to be `0`.
`<type>`	Optional Parameter: A value of `true` or `false` that indicates when payments are due. `True` indicates that payments are due at the beginning of the period (annuity due), `false` at the end (ordinary annuity). If `<type>` is omitted, it is assumed to be `false`.

EXAMPLES

```
-- Calculates amount of interest paid on a month's mortgage payment
x = ipmt(.07375/12,1.667*12,30*12,159000,0,false)
```

irr()

Financial Values

SYNTAX

```
irr(<values>[,<guess>])
```

DESCRIPTION A financial function that calculates the interest rate for a series of cash flow amounts (considered to occur at regular periods) and returns the internal rate of return. Payments out of pocket should be entered as negative values, while income should be entered as positive values.

PARAMETERS

PARAMETER	DESCRIPTION
<values>	A list of numbers that must contain at least one positive number (income) and one negative number (payment out of pocket). The order of this list indicates the sequence in which the cash flows occur. If a non-numeric value is found in the list, an error results and ToolBook displays the Execution Suspended message.
<guess>	Optional Parameter. A number between -0.1 and 0.1 that represents the expected internal rate of return on a cash flow. If omitted, it is assumed to be 0.1.

EXAMPLES

```
-- Calculates internal rate of return on the following series
x = irr("-35000,5000,8000,10000,12000,13500",0.2)
```

irregularPolygon

Object Type

SYNTAX

```
draw irregularPolygon from <location> to <location> to <location>...
```

DESCRIPTION

The object type name for an irregular polygon.

NOTES

An irregular polygon's parent can be the page, the background, or a group.

The bounds and vertices properties of an irregular polygon do not have the same values. An irregular polygon's bounds are the same as the location of the object's selection handles, while each vertex is a list of two numbers that give the location of the irregular polygon's vertices from the first angle drawn to the last angle drawn. ToolBook automatically closes an irregular polygon if its beginning and ending vertices are not the same values.

To reshape an irregular polygon, select it and send the reshape message, then drag the reshape handles to a new position.

To add vertices to an irregular polygon, select the irregular polygon and send the reshape message. Press the Shift key as you click a vertex handle; a new vertex handle appears adjacent to the existing handle. Then follow the same steps as you would to reshape the object.

To remove vertices from an irregular polygon, select the irregular polygon and send the reshape message. Press CTRL-Shift as you click a vertex handle. ToolBook reshapes the object without the selected vertex.

is

Logic Operators

SYNTAX

```
<expression> is <expression> [as <type>]
```

DESCRIPTION

Compares the expressions and returns true if the left expression and the right expression are equivalent. Returns false if the expressions are not equivalent.

The expressions can be compared as text, as numbers, as dates, or as names.

The = operator works identically to the is operator.

NOTES

Although the is operator works identically to the = operator you will find the is operator is not commonly used in this manner.

In OpenScript, string comparisons are case insensitive.

PARAMETERS

PARAMETER	DESCRIPTION
<expression>	Any expression that yields a numeric value or a value appropriate to the value of the <type> parameter.
<type>	number, text, date, or name

The default value of the <type> parameter is number, resulting in numeric comparison. If the <type> parameter is text, the two expressions are compared alphabetically, according to the ANSI values of the characters. If the <type> parameter is date, the expressions are compared as dates according to the current value of the sysDateFormat property.

EXAMPLES

```
while username is "admin"

if password is "12345"

put (age is 65) into checked of button "retire"
```

is in

Logic Operators

SYNTAX

```
<expression> is in <expression>
```

DESCRIPTION

Returns true if the left expression is found within the right expression. Otherwise returns false.

NOTES

This is a string comparison operator. If you attempt to use numbers, their string equivalents will be compared.

In OpenScript, string comparisons are case insensitive.

PARAMETERS

PARAMETER	DESCRIPTION
<expression>	Any expression.

EXAMPLES

```
if 123456 is in text of field "password"
    go to page "welcome"
end
```

is not

Logic Operators

SYNTAX

```
<expression> is not <expression> [as <type>]
```

DESCRIPTION

Compares the expressions and returns true if the left expression and the right expression are different. Returns false if the expressions are the same. The expressions can be compared as text, as numbers, as dates, or as names.

The <> operator works identically to the is not operator.

NOTES

In OpenScript, string comparisons are case insensitive.

PARAMETERS

PARAMETER	DESCRIPTION
<expression>	Any expression that yields a numeric value or a value appropriate to the value of the <type> parameter.
<type>	number, text, date, or name

The default value of the <type> parameter is number, resulting in numeric comparison. If the <type> parameter is text, the two expressions are compared alphabetically, according to the ANSI values of the characters. If the <type> parameter is date, the expressions are compared as dates according to the current value of the sysDateFormat property.

EXAMPLES

```
if password is not "12345"
    request "you don't have access to this file"
end
```

is not in

Logic Operators

SYNTAX

```
<expression> is not in <expression>
```

DESCRIPTION

Returns `true` if the left expression is not found within the right expression. Otherwise returns `false`.

NOTES

This is a string comparison operator. If you attempt to use numbers, their string equivalents will be compared.

In OpenScript, string comparisons are case insensitive.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	Any expression.

EXAMPLES

```
if 123456 is not in text of field "password"
    go to page "bad password"
end
```

isCDDrive()

TBDOS.DLL File Function (Drive)

SYNTAX

```
isCDDrive(<drive name>)
```

DESCRIPTION

Determines if a drive letter is a CD ROM Drive.

RETURNS

`0` The drive is not a valid CD drive.
`1` The drive is a valid CD drive.

NOTES

You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    INT isCDDrive(STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `isCDDrive32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<drive name>`	The letter designating the CD disk drive.

EXAMPLES

```
to handle buttonClick
    nam = name of this book
    driveLetter = char 1 of nam
    if isCDDrive(driveLetter) = 1
        -- don't try to save if this book is on a CD ROM
    else
        send save
    end
end
```

isCDDrive32()

TBFILE32.DLL File Function (Drive)

SYNTAX `isCDDrive32(<drive name>)`

DESCRIPTION Determines if a drive letter is a CD ROM Drive.

RETURNS
`0` The drive is not a valid CD drive.
`1` The drive is a valid CD drive.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    INT isCDDrive32(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<drive name>`	The letter designating the CD disk drive.

EXAMPLES

```
to handle buttonClick
    nam = name of this book
    driveLetter = char 1 of nam
    if isCDDrive32(driveLetter) = 1
        -- don't try to save if this book is on a CD ROM
    else
        send save
    end
end
```

isChanged

Property

DESCRIPTION A book property that specifies if changes have been made to the book since it was last saved.

NOTES You cannot set this property.

VALUES `True` or `false`.

If true, the book has changed since the last time it was saved.

EXAMPLES

```
to handle buttonClick
    if isChanged of this book = true
        request "Save this book now?" with "&Yes" or "&No"
        if IT is "Yes"
            send save
        end
    end
end
```

isItemInList()

UNDOCUMENTED String Functions

SYNTAX `isItemInList(<item>,<list>)`

DESCRIPTION Determines if the specified `<item>` exists in a comma separate `<list>` of items.

This function is equivalent to saying: `(itemOffset(x,y) <> 0)`

RETURNS The function returns `true` if the item is in the list, and `false` if the item is not in the list.

WARNING THIS IS AN UNDOCUMENTED FEATURE. What that means is that you can use it but you will most likely be unable to get assistance from Click2learn on how to use it, or troubleshoot it.

NOTES This function is essentially identical to the `ASYM_ItemInList()` TB89R.SBK function, however `isItemInList()` is written as a core function which makes it inherently faster to execute.

PARAMETERS

PARAMETER	DESCRIPTION
`<item>`	An expression that results in a string value.
`<list>`	An expression that results in a list.

EXAMPLES

```
to handle buttonClick
    possibleAnswers = "apple,banana,orange,pineapple"
    ask "What is one of my favorite fruits?"
    answ = IT
    if isItemInList(answ,possibleAnswers) = true
        request "You are correct."
    else
        request "Sorry, incorrect!"
    end
end
```

isNumber()

TB89R.SBK Validation

SYNTAX `isNumber(<value>)`

DESCRIPTION Checks if the specified value is a number.

This function works specifically with integer and decimal numbers.

It returns false for numbers that include special characters, such as $4.00.

RETURNS `True` if the value is a number.

`False` if not.

ACTIONS EDITOR The Actions Editor also supports this feature.

NOTES This function is provided by the TB89R.SBK. To successfully use this function in Runtime, you must first add the TB89R.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89R.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89R.SBK will already be there.

PARAMETERS

PARAMETER	DESCRIPTION
`<value>`	A literal numeric value or the name of a variable.

EXAMPLES

```
val = text of field "answer"
if isNumber(val)
    x = val * 1.5
end
```

isOpen

Property

DESCRIPTION A property of a viewer that specifies whether a viewer is open.

NOTES You can get but cannot set this property.

If a viewer is open but not shown, getting `isOpen` will still return `true`.

VALUES `True` or `false`.

If `true`, the viewer is currently open. If `false`, it viewer is not open.

EXAMPLES

```
if isOpen of viewer "help" = false
    show viewer "help"
end
```

isProperty()

UNDOCUMENTED Functions

SYNTAX `isProperty(<objRef>,<propName>)`

DESCRIPTION Determines if an object contains a specific property or user property.

RETURNS If the property exists the function returns `true`, otherwise returns `false`.

WARNING THIS IS AN UNDOCUMENTED FEATURE. What that means is that you can use it but you will most likely be unable to get assistance from Click2learn on how to use it, or troubleshoot it.

PARAMETERS

PARAMETER	DESCRIPTION
`<objRef>`	An expression that results in any object reference.
`<propName>`	An expression that results in a string, representing the property name you are looking for.

EXAMPLES

```
-- Check to see if a PartNumber property exists.
if isProperty(ellipse "parts bin","PartNumber") = true
    request "Yes, property exists."
end
```

isSharedScript()

UNDOCUMENTED Functions

SYNTAX `isProperty(<objRef>,<ssRef>)`

DESCRIPTION Determines if an object is configured to use a specifically named shared script resource.

RETURNS If the object is currently configured to use a shared script resource whose name matches the name passed to the function, the function returns `true`, otherwise returns `false`.

WARNING THIS IS AN UNDOCUMENTED FEATURE. What that means is that you can use it but you will most likely be unable to get assistance from Click2learn on how to use it, or troubleshoot it.

PARAMETERS

PARAMETER	DESCRIPTION
`<objRef>`	An expression that results in any object reference.
`<ssRef>`	An expression that results in a shared script name.

EXAMPLES

```
to handle buttonClick
    if isSharedScript(rectangle "TL31","autosizer") = true
        move rectangle "TL31" to 0,0
    end
end
```

isType()

Data Type Function

SYNTAX `isType(<type>,<container>,[<format list>])`

DESCRIPTION Tests whether a container's value includes one or more specified formats of a specified type. You must specify a value type and a list of one or more formats with which to test the specified value.

RETURNS If the value is in a valid format, the `isType()` function returns `true`. If the value is in an invalid format, the function returns `false`.

PARAMETERS

PARAMETER	DESCRIPTION
`<type>`	Any valid data type. If `word`, it must appear in quotes ("`word`").
`<container>`	A valid OpenScript container.
`<format list>`	A list of one or more valid format strings for the specified `<type>`. For example, if `<type>` is `date`, one or more date formats can be specified, such as "`mm/dd/y`" or "`mm/dd/yy,M/D/Y`". If `<format list>` is omitted, the function returns `true` if the value matches the appropriate system default format: `sysNumberFormat`, `sysDateFormat`, or `sysTimeFormat`.

EXAMPLES

```
-- isType returns true for this example
v1 = 5
get isType("word",v1)

-- isType returns false in this example because the date formats
-- do not match
vDate = "March 5, 1992"
get isType(date,vDate,"mm/dd/y")
```

IT

Special Variable

DESCRIPTION A special local variable that is automatically available in all handlers.

NOTES The value of `IT` does not persist between handlers. However, the value of `IT` does persist between statements executed in the Command window.

ToolBook automatically puts the value returned by an `ask`, `get`, `getRemote`,, readFile, `request`, `search`...`locateOnly`, or `seekFile` command into `IT`. If no destination container is provided with the `pop` command, its value is placed into `IT` as well.

Data put into `IT` by one command will be overwritten by data put into `IT` by any command that is executed later.

To maintain the value of `IT` past the next statement in a handler, put the value of `IT` into another variable.

EXAMPLES

```
ask "What is your name?"
request IT

while val <> null
    pop val
    request IT
end

get sortTextLines(var)
request IT
```

italic

Character Formatting

DESCRIPTION Sent to the current page when `italic` is chosen from the `Text` menu.

You can also `send` this message using the `send italic` statement. ToolBook's default response is to change the type style to italic or, if it's already italic, to turn off that style.

EXAMPLES

```
select chars 15 to 20 of text of field "list"
send italic
```

item, items

String Operators

SYNTAX

```
item <number> of <stringListRef>
items <number> to <number> of <stringListRef>
```

DESCRIPTION Used to refer to specific items in a text string. An `item` is defined as a sequence of characters surrounded by commas.

The first and last item in a string are the exceptions to this rule, as the first `item` will unlikely have a comma preceding it, as is the last `item` in a string unlikely to have a comma following it.

NOTES Various terms can be used with `item` including `first`, `last`, `middle`.

Only a comma is used as an item delimiter. It is not possible to tell ToolBook to use a different delimiter.

PARAMETERS

PARAMETER	DESCRIPTION
`<number>`	An expression that results in a number.
`<stringListRef>`	An expression that results in a string value.

EXAMPLES

```
xPos = item 1 of position of self
yPos = item 2 of position of self
```

```
item 2 of field "fruit" = "banana"
```

```
z = "This is item 1,This is item 2,This is item 3"
step k from 1 to itemCount(z)
    request item k of z
end
```

```
-- Position all objects on the current page to the 0,0 position
objs = objects of this page
step k from 1 to itemCount(objs)
    curObj = item k of objs
    position of curObj = 0,0
end
```

itemCount()

String Functions

SYNTAX

```
itemCount(<expression>)
```

DESCRIPTION Counts the number of item in an expression.

RETURNS Returns the number of items in a string. An `item` is defined as a sequence of characters surrounded by commas.

The first and last item in a string are the exceptions to this rule, as obviously the first item will unlikely have a comma preceding it, as is the last item in a string unlikely to have a comma following it.

NOTES You can also count characters with `charCount()` and words with `wordCount()`.

If you need to know how many items are in a `stack`, it is not recommended that you use `itemCount()`. This is because a stack `data type` can hold more than `64k` of data whereas a standard string can only hold a maximum of `64k`. This `itemCount()` function is a String function, which means it can only handle strings up to `64k` in size, and will produce an `Execution Suspend` error if you attempt to exceed it. Since ToolBook lacks a `stackItemCount()` function, if you need to know how many items are in a stack, use this technique:

```
-- count items in the stack variable var
tmpStack = var
ct = 0
while tmpStack <> null
    pop tmpStack
    increment ct
end
request "There are" && ct && "items in your stack."
```

PARAMETERS

PARAMETER	DESCRIPTION
<expression>	An expression that results in a string value.

EXAMPLES

```
to handle buttonClick
    x = "cat,dog,lion"
    step k from 1 to itemCount(x)
        curItem = item k of x
        request "Item" && k && "=" && curItem
    end
end
```

itemOffset()

TBDLG.DLL String Functions

SYNTAX `itemOffset(<item>,<list>)`

DESCRIPTION Locates the first occurrence of an item in a list.

RETURNS Returns the item position in which `<item>` was found in `<list>`. If `<item>` cannot be found within `<list>` then `0` is returned.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    INT itemOffset(STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<item>	An expression that results in a string value.
<list>	An expression that results in a list.

EXAMPLES

```
to handle buttonClick
    lst = "apple,banana,orange,pineapple"
    ask "What are you looking for?"
    answ = IT
    pos = itemOffset(answ,lst)
    if pos > 0
        request "Yes, that item is in position" && pos
    else
        request "Sorry, didn't find a match"
    end
end
```

itemSelected

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of a combobox, list box or radio button group specifying an Array - where each element is `true` or `false` depending on whether or not that item is selected.

NOTES You can get or set this property in the Actions Editor.

Note that the array indexing is `0` based not `1` based. This means that if you are trying to query the first textline, you would need to query the index [`0`].

This property has no real meaning to OpenScript and should be used only within the Actions Editor.

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

VALUES `True` if the item is selected, otherwise `false`.

EXAMPLES

```
On click...
  Set val to itemSelected of field "groceries"
  Display alert: val[6]
```

itemText

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of a combobox, list box or radio button group specifying an Array - where each element contains the text (or caption) of each item.

NOTES You can get or set this property in the Actions Editor.

Note that the array indexing is `0` based not `1` based. This means that if you are trying to query the first textline, you would need to query the index [`0`].

This property has no real meaning to OpenScript and should be used only within the Actions Editor.

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

EXAMPLES

```
On click...
  Set val to itemText of field "groceries"
  Display alert: val[0]
```

keepMenuBar

Property

DESCRIPTION A property of the book that specifies how ToolBook manages the Main window's `Reader`-level menu bar during navigation from one book to another.

NOTES You can get or set this property.

When you navigate to a different book in the Main window, ToolBook displays the menu bar assigned to the Main window in the new book. Set `keepMenuBar` to `true` to continue displaying the menu bar from the previous book.

To ensure that ToolBook loads the Main window's menu bar regardless of the setting for a previous book's `keepMenuBar` property, add a `restore menuBar at Reader` statement to a handler for the `enterApplication` message in a book's script.

VALUES `True` or `false`.

The default is `false` when you create a new book.

EXAMPLES

```
keepMenuBar of this book = true
```

Key constants and related values

1	keyLeftButton	39	keyRightArrow	76	keyL	111	keyDivide
2	keyRightButton	40	keyDownArrow	77	keyM	112	keyF1
3	keyCancel	41	keySelect	78	keyN	113	keyF2
4	keyMiddleButton	42	keyPrint	79	keyO	114	keyF3
8	keyBack	43	keyExecute	80	keyP	115	keyF4
9	keyTab	44	keyCopy	81	keyQ	116	keyF5
12	keyClear	45	keyInsert	82	keyR	117	keyF6
13	keyEnter	46	keyDelete	83	keyS	118	keyF7
16	keyShift	47	keyHelp	84	keyT	119	keyF8
17	keyControl	48	key0	85	keyU	120	keyF9
18	keyMenu	49	key1	86	keyV	121	keyF10
19	keyPause	50	key2	87	keyW	122	keyF11
20	keyCapital	51	key3	88	keyX	123	keyF12
21	keyKana	52	key4	89	keyY	124	keyF13
22	keyRomanji	53	key5	90	keyZ	125	keyF14
23	keyZenkaku	54	key6	96	keyNumpad0	126	keyF15
24	keyHiraGana	55	key7	97	keyNumpad1	127	keyF16
25	keyKanji	56	key8	98	keyNumpad2	144	keyNumLock
27	keyEscape	57	key9	99	keyNumpad3	145	keyScrollLock
28	keyConvert	65	keyA	100	keyNumpad4	186	keySemicolon
29	keyNonConvert	66	keyB	101	keyNumpad5	187	keyEqual
30	keyAccept	67	keyC	102	keyNumpad6	188	keyComma
31	keyModeChange	68	keyD	103	keyNumpad7	190	keyPoint
32	keySpace	69	keyE	104	keyNumpad8	191	keySlash
33	keyPrior	70	keyF	105	keyNumpad9	192	keyBackQuote
34	keyNext	71	keyG	106	keyMultiply	219	keyLeftBracket
35	keyEnd	72	keyH	107	keyAdd	220	keyBackslash
36	keyHome	73	keyI	108	keySeparator	221	keyRightBracket
37	keyLeftArrow	74	keyJ	109	keySubtract	222	keyQuote
38	keyUpArrow	75	keyK	110	keyDecimal		

Keyboard equivalents for selected key constants

keyClear	5 on numeric keypad when NumLock is off
keyMenu	Alt
keyCapital	CapsLock (down means on, up means off)
keyPrior	PgUp
keyNext	PgDn
keyMultiply	* on numeric keypad
keyAdd	+ on numeric keypad
keySubtract	– on numeric keypad
keyDivide	/ on numeric keypad
keyDecimal	. on numeric keypad when NumLock is on

keyColor

Property

DESCRIPTION A property of a bitmap resource that specifies what color in the bitmap will be treated as transparent.

NOTES This property is used in conjunction with the bitmap's `useChromakey` property. If the `useChromakey` property is `true`, the portions of the bitmap that match the value of the `keyColor` property become transparent.

For paint objects, the `fillColor` property specifies the color that is masked as transparent; for bitmap resources, the `keyColor` property performs this function.

For both paint objects and resources, the useChromakey property must be true for the transparency to take effect.

VALUES A list of three non-negative numbers representing the `RGB` color value, or a valid color constant.

EXAMPLES

```
keyColor of bitmap "car door" = red
```

keyChar

Keyboard Event Message

SYNTAX

```
keyChar <key>,<isShift>,<isCtrl>
```

DESCRIPTION Sent to the object with the focus or, if there is no focus, to the current page when a key is pressed at `Reader` level.

The ToolBook default response depends on the `key` and the context in which it is pressed.

For example, if the focus is in a field, the default response for alphanumeric keys is to insert the corresponding character into the field.

NOTES A `keyChar` message is not sent when the user releases an `accelerator key` (for example, `F3`) or a key that does not generate a character that can be displayed (for example, the arrow keys). To control response to these actions in an application, write handlers for the related events. For example, to control the response when a user presses `F3`, write a handler for the `Reader` or `Author` menu event message. To control the response when a user presses a navigation key, write a `keyDown` handler.

The `keyChar` message differs from the `keyDown` and `keyUp` messages in that the `<key>` parameter for `keyChar` is the `ANSI` value of the typed character.

PARAMETERS

PARAMETER	DESCRIPTION
`<key>`	The `ANSI` value of a typed character. For a list of constants, see the `ANSI` character table in the online help.
`<isShift>`	`True` or `false`, indicating whether the `Shift` key is pressed in conjunction with the key specified by `<key>`.
`<isCtrl>`	`True` or `false`, indicating whether the `Ctrl` key is pressed in conjunction with the key specified by `<key>`.

EXAMPLES

```
-- Validates that keyboard input is numeric (such as 0 - 9)
to handle keyChar key
    if key < 48 or key > 57
        -- Key is not allowed. Brek out of this handler
        -- and do not forward the message
        break
    else
        forward
    end
end
```

keyDown

Keyboard Event Message

SYNTAX `keyDown <key>,<isShift>,<isCtrl>`

DESCRIPTION Sent to the object with the focus or, if there is no focus, to the current page when a key is pressed at `Reader` level.

The ToolBook default response depends on the `key` and the context in which it is pressed.

For example, if the focus is in a field, ToolBook's default response for navigation keys and editing keys is to perform the respective navigation or editing action.

NOTES A `keyDown` message is not sent when the user presses an accelerator key combination (for example, `Alt+Right Arrow` for `next`). To control response to these actions in an application, write handlers for the related events. For example, to control the response when a user presses `Alt+Right Arrow`, write a handler for the `next` event message.

If the menu item associated with this shortcut key is removed, the `keyDown` menu event message is sent instead of the standard menu item message.

PARAMETERS

PARAMETER	DESCRIPTION
`<key>`	An integer or key constant that represents a key. For a list of constants, see Key constants (table) in the online Help.
`<isShift>`	`True` or `false`, indicating whether the `Shift` key is pressed in conjunction with the key specified by `<key>`.
`<isCtrl>`	`True` or `false`, indicating whether the `Ctrl` key is pressed in conjunction with the key specified by `<key>`.

EXAMPLES

```
to handle keyDown key
    if key = keyEscape
        send exit
    else
        forward
    end
end
```

keyMnemonic

Keyboard Event Message

SYNTAX `keyMnemonic <key>`

DESCRIPTION Sent when the user presses a mnemonic access character that is not currently defined for a ToolBook menu or button.

NOTES This message is sent to the page displayed in the topmost target window; that is, if the focus is in a child window, the message is sent to that window's parent window.

A `mnemonic` key is pressed in conjunction with the `Alt` key. The ToolBook menus include `mnemonic` access keys that are either built-in or defined when you name a menu or button and place an ampersand (`&`) in the name before the character that will serve as the access key. Use the `keyMnemonic` message to define back door keys that aren't currently defined for any menus or buttons in an application.

You can use the `keyMnemonic` message in a script to customize keyboard access to an application. For example, you might write a handler for the `keyMnemonic` message that controls the volume in a multimedia application. The user could press `Alt` plus a number key, with `0` as no volume and `9` as full volume.

PARAMETERS

PARAMETER	DESCRIPTION
<key>	An integer or key constant that represents a key. For a list of constants, see Key constants (table) in the online Help.

EXAMPLES

```
-- Creates keyboard access to volume control by handling
-- the keyMnemonic message
to handle keyMnemonic keyCode
    system svVolume
    if keyCode >= key0 and keyCode <= key9
        set svVolume to keyCode - key0
    end
end
```

keyState()

Keyboard Function

SYNTAX

```
keyState(<key>)
```

DESCRIPTION

Indicates the `up` or `down` state of a keyboard key.

RETURNS

Returns `down` if the specified key is pressed. Returns `up` if the specified key is not currently pressed.

PARAMETERS

PARAMETER	DESCRIPTION
<key>	An integer or `key constant` that represents a key. For a list of constants, see the Key Constants table in the online help.

EXAMPLES

```
to handle buttonClick
    step k from 1 to pageCount of this book
        curPage = page k of this book
        send printMyPageStatusDetails(curPage)
        -- allow user to break out by using ESC key
        if keyState(keyEscape) = "down"
            break buttonClick
        end
    end
end
```

keyUp

Keyboard Event Message

SYNTAX

```
keyUp <key>,<isShift>,<isCtrl>
```

DESCRIPTION

Sent to the object with the focus or, if there is no focus, to the current page when a key is released at `Reader` level.

The ToolBook default response depends on the `key` and the context in which it is released.

For example, if the focus is in a field, ToolBook's default response for navigation keys and editing keys is to perform the respective navigation or editing action.

NOTES

A `keyUp` message is not sent when the user presses an accelerator key combination (for example, `Alt+Right Arrow` for `next`). To control response to these actions in an application, write handlers for the related events. For example, to control the response when a user presses `Alt+Right Arrow`, write a handler for the `next` event message.

If the menu item associated with this shortcut key is removed, the `keyUp` menu event message is sent instead of the standard menu item message.

PARAMETERS

PARAMETER	DESCRIPTION
<key>	An integer or key constant that represents a key. For a list of constants, see Key constants (table) in the online Help.
<isShift>	True or false, indicating whether the Shift key is pressed in conjunction with the key specified by <key>.
<isCtrl>	True or false, indicating whether the Ctrl key is pressed in conjunction with the key specified by <key>.

EXAMPLES

```
to handle keyUp key
    if key = keyEscape
        send exit
    else
        forward
    end
end
```

lastScore

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION

An Actions Editor provided property of a question object which specifies the score of the question as of the last time it was scored.

NOTES

You can get but not set this property in OpenScript as well as in the Actions Editor.

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

EXAMPLES

```
text of field "Score" = lastScore of group "multiple choice"
```

layer

Property

DESCRIPTION

A recordfield, field, group, button, combobox, draw object, ole object, paint object, picture object or stage property that specifies the object's layer number.

NOTES

You can get or set this property. The layer order is the stacking order of objects. An object with a lower layer number will appear under an object with a higher layer number. As you add objects to your page, they will appear on the next highest layer number.

The layer order will always start from 1 and progress higher one by one. ToolBook will renumber the layer properties of object as needed. For example if you have 3 objects on layers 1, 2 and 3, and you delete the object on layer 2 - the object on layer 3 moves to layer 2. If you reassign a object's layer value from 3 to 1, the object will move to layer 1, the object on layer 1 will move to layer 2 and the object on layer 2 will move to layer 3.

To move an object to the highest layer number, set its layer value to 0. ToolBook will determine what the highest layer is and automatically assign it for you. To move an move to the lowest layer, set its layer value to 1.

Tab Order is determined by layer order. The tab order begins at the lowest layer number (1) and continues to the highest layer number.

VALUES

The layer number of the object.

EXAMPLES

```
-- if a user clicks on this object, bring it to the top layer
to handle buttonClick
    layer of target = 0
end

-- move the flower on top of the stem
layer of paintObject "flower" = layer of paintObject "stem" + 1
```

leaveApplication

Event Message

DESCRIPTION Sent to the current page of the Main window when a book closes in the ToolBook Main window.

NOTES The `leaveApplication` message is sent just after the `leaveBook` message.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle leaveApplication
    -- close open file and media before exiting
    SYSTEM fName
    mmStop all
    mmClose all
    close fName
    forward
end
```

leaveBackground

Event Message

DESCRIPTION Sent to the current page when the user closes the book displaying the current background, closes a viewer displaying the current background, or goes to a page with a different background.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle leaveBackground
    sysLockScreen = true
    show group "infoBox"
    hide group "helpBox"
    sysLockScreen = false
    forward
end
```

leaveBook

Event Message

DESCRIPTION Sent to the current page when the user exits ToolBook, goes to another book, or closes the last viewer displaying a page from the book.

NOTES To close a book, send the `exit` message.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle leaveBook
    closeFile "userlog.txt"
    forward
end
```

leaveButton

Event Message

DESCRIPTION Sent to the `button` that has the focus just before another object gets the focus.

NOTES To ensure that a button does not have the focus, set `focus` to `null`.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle leaveButton
    fillColor of self = gray
    forward
end

to handle leaveButton
    fillColor of self = lightGray
    forward
end
```

leaveComboBox

Event Message

DESCRIPTION Sent to a `combobox` that has the focus just before another object gets the focus.

NOTES To ensure that a `combobox` does not have the focus, set `focus` to `null`.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle leaveComboBox
    clear text of self
    forward
end
```

leaveDropDown

Event Message

DESCRIPTION Sent to a `combobox` when its drop-down list box closes.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
-- Place this handler in the combobox's script to set the contents
-- of the combobox before the drop-down list box is displayed,
-- then clear the contents when the list box closes
to handle enterDropDown
    my dropDownItems = "Item 1" & CRLF & "Item 2" & CRLF & "Item 3"
    forward
end

to handle leaveDropDown
    my dropDownItems = null
    forward
end
```

leaveField

Event Message

DESCRIPTION Sent to the `field` that has the `focus` just before another object gets the `focus`.

NOTES To ensure that a `field` does not have the `focus`, set focus to `null`.

Handlers can be written for the `leaveField` message to handle data validation.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle leaveField
    txt = my text
    txt = ASYM_Trim(txt)
    txt = upperCase(txt)
    my text = txt
    forward
end

-- same as above but more compact
to handle leaveField
    my text = upperCase(ASYM_Trim(text of self))
    forward
end
```

leavePage

Event Message

DESCRIPTION Sent to the page just before another page gets the focus.

NOTE This message can be sent as the result of page navigation, or when a viewer closes.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle leavePage
    position of paintObject "apple" = 0,0
    text of field "comment" = null
    forward
end
```

leaveRecordField

Event Message

DESCRIPTION Sent to the `recordfield` that has the focus just before another object gets the focus.

NOTE To ensure that a `recordfield` does not have the focus, set `focus` to `null`.

Handlers can be written for the `leaveRecordField` message to handle data validation.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle leaveRecordField
    my text = upperCase(ASYM_Trim(text of self))
    forward
end
```

leaveSystem

Event Message

DESCRIPTION Sent to the current page of the Main window just before the instance of ToolBook is closed.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
-- before closing down, close DDE session opened earlier
to handle leaveSystem
    executeRemote "send exit" application "ToolBook" topic \
      "tutorial.tbk"
    forward
end
```

leaveWindow

Event Message

SYNTAX `leaveWindow <new viewer>,<new handle>`

DESCRIPTION Sent to a `viewer` when the user switches to a different application or clicks a different viewer in the same application, or when a viewer is `closed` or `hidden`.

NOTE When `leaveWindow` is sent, the viewer is no longer the active window on the desktop, and the value for `focusWindow` is set to another window.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
`<new viewer>`	A valid reference to the viewer that is gaining the focus. If the window that is receiving the focus is not a ToolBook viewer, <new viewer> is `null`.
`<new handle>`	The window handle of the viewer that is receiving the focus.

EXAMPLES

```
to handle leaveWindow
    hide group "selectionRectangle"
    forward
end
```

left

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of almost all ToolBook object types that specifies the current left position of that object.

NOTES You can get or set this property in OpenScript as well as in the Actions Editor.

In OpenScript, this can also be achieved with: `item 1 of position of <target>`

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

VALUES A whole numbers in `page units`.

EXAMPLES

```
left of button "foo" = 500
```

```
x = left of button "foo"
```

leftQuote

Special Constants

DESCRIPTION Represents the ANSI value of 171.

Using a standard font, that character would look like this: «

You might expect it to look like a quotation mark of some sort but it is instead a left chevron character.

EXAMPLES

```
x = leftQuote & "Apple" & rightQuote
```

leftString()

Nonexistent String Functions

SYNTAX

```
leftString(<string>,<num chars>)
```

DESCRIPTION Extracts a number of characters from the left (beginning) of a string.

RETURNS Returns the first <num chars> characters of the string.

NOTES Although this function does not exist in ToolBook you can add it by putting this script in the script of your book at which point you can call this function from anywhere in that book.

```
to get leftString str, num
    return chars 1 to num of str
end
```

Also see rightString() and midString().

PARAMETERS

PARAMETER	DESCRIPTION
<string>	An expression that results in a string value.
<num chars>	An expression that results in a whole number.

EXAMPLES

```
-- First 6 chars of a part number is the bin location.
-- Find the bin number for this part.
to handle buttonClick
    partNum = textline 44 of field "allParts"
    binNumber = leftString(partNum,6)
    put binNumber into text of field "bin"
end
```

(Linefeed) LF

Spacing Constants

DESCRIPTION Represents the linefeed ANSI character. This is equivalent to ANSI 10. In ToolBook an LF by itself is not of much use, but is when used as CRLF.

If it possible to generate an LF in a text field when you are typing if you press SHIFT-Enter on the keyboard. A normal Enter would generate CRLF.

EXAMPLES

```
if text of field "story" contains LF
```

lightGray

Color Constants

DESCRIPTION Represents the color `lightGray`. This is equivalent to the `RGB` value of `192,192,192` and the `HLS` value of `0,75,0`.

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES

```
rgbFill of field "list" = lightGray
```

```
if rgbStroke of rectangle "drop area" = lightGray
    rgbStroke of rectangle "drop area" = 0,191,0
end
```

```
strokeColor of ellipse id 14 = lightGray
```

line

Object Type

SYNTAX

```
draw line from <location> to <location>
```

DESCRIPTION The object type name for a `line`.

NOTES A line's parent can be the page, the background, or a group if the line is in a group.

The `bounds` for a line without line ends is a list of two pairs of numbers that represent the end points of the line. For lines with line ends, `bounds` refers to the smallest rectangle that encompasses both the line and its line end or ends.

lineCount

Property

DESCRIPTION A combobox property that specifies the number of lines displayed in the dropdown list box.

NOTES You can get or set this property.

The number of lines determines the textline height of the dropdown list box.

When you reset this value, ToolBook resizes the dropdown list box.

VALUES A positive whole number that specifies the number of lines displayed in the dropdown list box.

EXAMPLES

```
lineCount of combobox "partList" = 4
```

```
-- if more than 8 item in dropdown items set height to 8 otherwise
-- set height to the same value of the textlineCount
obj = combobox "partList"
dropDownItems of obj = lst
lineCount of obj = min(8,textlineCount(obj))
```

lineEndSize

Property

DESCRIPTION A property of a `line`, `angled line`, `arc`, or `curve` that specifies the size of the line ends for that object.

NOTES You can get or set this property.

VALUES A list of two integers from `1` to `9`, indicating size. `1` is the minimum, `9` is the maximum.

The first value in the list specifies the size for the starting point of the line; the second value specifies the size for the end point of the line. The default is the value of `sysLineEndSize`.

EXAMPLES

```
lineEndSize of line "Pointer" = 3,3
```

lineEndsPalette

System Object

DESCRIPTION The object type name for the `line ends palette`, which is used to specify the `lineEndStyle` and `lineEndSize` properties for lines, angled lines, arcs, and curves.

NOTES To open the line ends palette, use the `show lineEndsPalette` statement in a script.

You can also show or hide the line ends palette by pressing Ctrl and clicking the Line Ends palette button on the tool bar.

The line ends palette cannot be shown in Runtime ToolBook.

Use the `hide`, `show`, and `move` commands to control the visibility and position of the line ends palette.

PROPERTY	VALUES
`bounds`	List of four whole numbers in `pixels`.
`position`	List of two whole numbers in `pixels`.
`vertices`	List of four whole numbers in `pixels`.
`visible`	`True` or `false`.

lineEndStyle

Property

DESCRIPTION A property of a line, angled line, arc, or curve that specifies a line end style for the line ends of that object.

NOTES You can get or set this property.

VALUES A list of two of the following values: `filledHead`, `filledTail`, `openHead`, `openTail`, `solidHead`, `solidTail`, or `none`.

The first value in the list specifies the style for the starting point of the line; the second value specifies the style for the end point of the line.

The default is the value of `sysLineEndStyle`.

EXAMPLES

```
lineEndStyle of line "Pointer" = "openTail,openHead"
```

linePalette

System Object

DESCRIPTION The object type name for the `line palette`, which is used to specify the borders around graphic objects or the line styles of draw objects with open end points.

NOTES To open the line palette, use the `show linePalette` statement in a script. You can also show or hide the line palette by clicking the Line palette button on the tool bar.

The line palette cannot be shown in Runtime ToolBook.

Use the `hide`, `show`, and `move` commands to control the visibility and position of the line palette.

PROPERTY	VALUES
`bounds`	List of four whole numbers in `pixels`.
`position`	List of two whole numbers in `pixels`.
`vertices`	List of four whole numbers in `pixels`.
`visible`	`True` or `false`.

lineStyle

Property

DESCRIPTION A property that specifies the line style of draw objects.

A property that specifies the line style around picture and paint objects.

NOTES You can get or set this property.

VALUES The following are valid values for lineStyle: `Dashed`, `dotted`, `none (0)`, or `1`, `2`, `3`, `4`, `6`, or `8`.

The values `0 - 8` represent the thickness of the line. The default is the value of `sysLineStyle` when the object was created.

EXAMPLES

```
lineStyle of ellipse "mouse" = "dotted"
```

linkDLL

Control Structure

SYNTAX

```
linkDLL <DLL name>
    <return type> <function name><(parameter type list)>
    [...]
end [linkDLL]

linkDLL <DLL name>
    <return type> <alias> = <function name><(parameter type list)>
    [...]
end [linkDLL]

linkDLL <DLL name>
    <return type> <alias> = <ordinal><(parameter type list)>
    [...]
end [linkDLL]
```

DESCRIPTION Links the specified `16-bit` DLL functions so that they can be called from a script.

NOTES If a DLL function is declared incorrectly, errors such as incorrect return values, General Protection Faults (`GPFs`), and stack errors can result. If you get this type of error, double-check the declaration of the DLL function to ensure that you use the correct ToolBook data type for the parameter `type list` and `return type`.

DATA TYPES FOR DLL FUNCTION PARAMETERS		
Windows C data type	**linkDLL parameter type**	**linkDLL return type**
`int`	`INT`	`INT`
`BOOL`	`INT`	`INT`
`void`	`none`	`INT`
`unsigned char, BYTE`	`BYTE`	`BYTE`
`UINT, unsigned int, word`	`WORD`	`WORD`
`HANDLE, HWND, ETC`	`WORD`	`WORD`
`long`	`LONG`	`LONG`
`unsigned long`	`DWORD`	`DWORD`
`FLOAT`	`FLOAT`	`FLOAT`
`DOUBLE`	`DOUBLE`	`DOUBLE`

STRING DATA TYPES FOR DLL FUNCTION PARAMETERS		
Windows C data type	**linkDLL parameter type**	**linkDLL return type**
`VOID FAR *, CHAR FAR *, LPSTR, LPCSTR`	`STRING or POINTER`	`POINTER`
`DWORD*`		`STRING*`

* Returns a handle to the string in the LOWORD and a value for sysErrorNumber in the HIWORD. If no error occurs, the HIWORD is NULL; ToolBook locks the string handle returned in the LOWORD, retrieves its value, and unlocks and frees the handle. If a null string is returned and there is no error, the LOWORD still contains a handle for a zero-terminated string of length zero. If there is an error, the HIWORD is set to any value other than NULL; ToolBook ignores the LOWORD, returns a null string to OpenScript, and sets sysErrorNumber to the value stored in the HIWORD.

PARAMETERS

PARAMETER	DESCRIPTION
`<dll name>`	The name (and path if necessary) of the DLL file with the functions that are to be made available to ToolBook. If you're using one of the built-in libraries for Windows, specify the module name rather than the file name. This ensures that your scripts perform properly on systems in which these standard modules may have different file names. Module names are `KERNEL`, `USER`, `GDI`, and `DISPLAY` (not `KRNL386.EXE`, `USER.EXE`, `GDI.EXE`, or `WNSPD800.DRV`).
`<return type>`	The literal data type that the function returns. The allowed values are: `BYTE`, `DOUBLE`, `DWORD`, `FLOAT`, `INT`, `LONG`, `POINTER`, `STRING`, or `WORD`.
`<alias>`	An alternate name (an alias) for the function. For example, if the literal name of the function in the DLL is `Save`, an `alias` such as `mySave` can be assigned to avoid a conflict with the OpenScript keyword `save`.
`<ordinal>`	The ordinal reference to the DLL function name.
`<function name>`	The literal DLL function name.
`<parameter type list>`	A comma separated list of one or more data types for the function's parameter or parameters. The allowed values are: `BYTE`, `DOUBLE`, `DWORD`, `FLOAT`, `INT`, `LONG`, `POINTER`, `STRING`, or `WORD`.

EXAMPLES

```
to handle linkDLLs
    linkDLL "tbdos.dll"
        -- Alias the fileExists function
        INT doesTheFileExist = fileExists(STRING)
        STRING getFileList(STRING)
    end
    linkDLL "user"
        INT showWindow(WORD,INT)
        WORD setActiveWindow(WORD)
    end
end
```

SYNTAX

```
linkDLL32 <DLL name>
    <return type> <function name><(parameter type list)>
    [...]
end [linkDLL]

linkDLL32 <DLL name>
    <return type> <alias> = <function name><(parameter type list)>
    [...]
end [linkDLL]

linkDLL32 <DLL name>
    <return type> <alias> = <ordinal><(parameter type list)>
    [...]
end [linkDLL]
```

DESCRIPTION

Links the specified 32-bit DLL functions so that they can be called from a script.

NOTES

If a DLL function is declared incorrectly, errors such as incorrect return values, General Protection Faults (GPFs), and stack errors can result. If you get this type of error, double-check the declaration of the DLL function to ensure that you use the correct ToolBook data type for the parameter `type list` and `return type`.

32-bit API functions with the same name as a 16-bit API function such as `FindWindow()` need to be declared with an "A" after the function name, for example:

```
linkDLL32
    WORD FindWindowA(STRING,STRING)
end
```

DATA TYPES FOR DLL FUNCTION PARAMETERS		
Windows C data type	**linkDLL parameter type**	**linkDLL return type**
int	INT	INT
BOOL	INT	INT
void	none	INT
unsigned char, BYTE	BYTE	BYTE
UINT, unsigned int, word	WORD	WORD
HANDLE, HWND, ETC	WORD	WORD
long	LONG	LONG
unsigned long	DWORD	DWORD
FLOAT	FLOAT	FLOAT
DOUBLE	DOUBLE	DOUBLE

STRING DATA TYPES FOR DLL FUNCTION PARAMETERS		
Windows C data type	**linkDLL parameter type**	**linkDLL return type**
VOID FAR *, CHAR FAR *, LPSTR, LPCSTR	STRING or POINTER32	STRING or POINTER32
DWORD*		STRING*

* Returns a handle to the string in the LOWORD and a value for sysErrorNumber in the HIWORD. If no error occurs, the HIWORD is NULL; ToolBook locks the string handle returned in the LOWORD, retrieves its value, and unlocks and frees the handle. If a null string is returned and there is no error, the LOWORD still contains a handle for a zero-terminated string of length zero. If there is an error, the HIWORD is set to any value other than NULL; ToolBook ignores the LOWORD, returns a null string to OpenScript, and sets sysErrorNumber to the value stored in the HIWORD.

PARAMETERS

PARAMETER	DESCRIPTION
`<dll name>`	The name (and path if necessary) of the DLL file with the functions that are to be made available to ToolBook.
`<return type>`	The literal data type that the function returns. The allowed values are: `BYTE`, `DOUBLE`, `DWORD`, `FLOAT`, `INT`, `LONG`, `POINTER32`, `STRING`, or `WORD`.
`<alias>`	An alternate name (an alias) for the function. For example, if the literal name of the function in the DLL is `Save`, an `alias` such as `mySave` can be assigned to avoid a conflict with the OpenScript keyword `save`.
`<ordinal>`	The ordinal reference to the DLL function name.
`<function name>`	The literal DLL function name. Note that `32-bit` function names are `case sensitive`.
`<parameter type list>`	A comma separated list of one or more data types for the function's parameter or parameters. The allowed values are: `BYTE`, `DOUBLE`, `DWORD`, `FLOAT`, `INT`, `LONG`, `POINTER32`, `STRING`, or `WORD`.

EXAMPLES

```
to handle linkDLLs
    linkDLL32 "Kernel32.dll"
        INT messageBox = MessageBoxA (DWORD,STRING,STRING,DWORD)
    end
end
```

linkSysBook

Notification Message

DESCRIPTION

Sent when a system book is `linked` to a book as the result of being added to a book's `sysBooks` property.

This message is sent to the system book that is linked, which becomes the value of `target`.

NOTE

To prevent a system book from responding to this message multiple times when other system books exist, be sure to handle this message in *each* system book; do not `forward` it.

Also, check the value of the `target` property to ensure that a handler is handling the message intended for a particular system book.

EXAMPLES

```
to handle linkSysBook
    if target = self
        send linkAlldlls
    end
end
```

listToTextline()

TBDLG.DLL String Functions

SYNTAX

`listToTextline(<list>)`

DESCRIPTION

This function converts your string of items into a string of textlines.

RETURNS

This conversion occurs by replacing comma characters in a string with `CRLF`. Note that a comma contained in a set of `quotes` will not be replaced.

Example: `a,b,c,"d,e,f"` would result in only `4` textlines rather than `6`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING listToTextline(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<list>	An expression that results in a string value.

EXAMPLES

```
to handle buttonClick
    -- show system book stack as textlines rather than a list
    request listToTextline(sysBooks)
end

to handle buttonClick
    -- show a list of objects on the page
    request listToTextline(objects of this page)
end
```

ln()

Logarithmic Values

SYNTAX

```
ln(<number>)
```

DESCRIPTION Returns the natural logarithm of a number. Natural logarithms are based on the constant e (2.71828182845904).

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<number>	An expression that yields a number.

EXAMPLES

```
x = ln(86)          -- equals 4.454347
x = ln(2.7182818)   -- equals 1
x = ln(exp(3))      -- equals 3
x = exp(ln(4))      -- equals 4
```

local

Variable Command

SYNTAX

```
local [<type>] <variable list>
local [<type>] [fixed|dynamic] <variable name>[<size>][<size>]...
```

DESCRIPTION Declares a variable name and makes its contents available within the context of the current handler. ToolBook considers all undeclared variables to be `local` variables. If it doesn't recognize the specified variable, ToolBook creates a new `local` variable with that name and puts the value into that variable.

You have the option to declare data types for `local` and `system` variables and arrays.

ToolBook initializes the value of a non-typed `local` variable to `null` when it is declared. ToolBook initializes the value of a `local` variable according to its data type.

DATA TYPE	VALUE
DWORD, INT, LONG, REAL, WORD	`0`
LOGICAL	`false`
POINT	`0,0`
COLOR	`0,0,0`
STACK, STRING	`null`
all other types	`null`

The value of a `local` variable is known to ToolBook only within the handler in which it is defined and persists only during execution of that handler. However, you can send a `local` variable's value to another handler by sending it as a message parameter.

When you assign a data type to a `local` variable, ToolBook converts the value to the declared type. Any reference to the variable assumes the value is of the declared type. Once y ou assign a type to a variable, you cannot use it as a different type.

The following operations function most efficiently when you use the suggested data type:

USE	DATA TYPE
Numeric operations	`REAL`
Text operations	`STRING`
Logical operations	`LOGICAL`
Step loops	`LONG`

PARAMETERS

PARAMETER	DESCRIPTION
`<variable list>`	One or more variable names.
`<type>`	The name of the data type.
`fixed | dynamic`	Declares a local array as either fixed or dynamic. A fixed array cannot be reset or expanded.
`<size>`	The number of elements in a dimension of a local array. Note that ToolBook is limited to a 16 dimensional array, as in: `var[][][][][][][][][][][][][][][]`

EXAMPLES

```
local vCount
vCount = 27

-- Local variables declared with a data type
local LOGICAL dialogIsUp
local LOGICAL StepCount

-- Local fixed arrays declared with a data type
local WORD fixed myArray[4][5]
local DATE fixed ToDo[100]
```

locked

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of a question object which specifies if the question is currently locked.

NOTES You can get but not set this property in OpenScript as well as in the Actions Editor.

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

VALUES `True` if the question has been locked.

EXAMPLES
```
isLocked = locked of group "multiple choice"
```

lockScreen

Property

DESCRIPTION A property of a viewer that specifies whether ToolBook allows screen updates to the contents of the viewer during the execution of the current handler.

NOTES You can get or set this property.

When `lockScreen` is set to `true` in a handler, it is automatically reset to `false` when ToolBook finishes executing the handler.

If you set `lockScreen` to `true` in a viewer's script, you can make several changes to the contents of a viewer, then show all the changes at once, instead of allowing ToolBook to update the screen after each change.

VALUES `True` or `false`.

The default is `false`.

If `false`, the screen will update as needed.

If `true`, the screen will update only after the screen is unlocked, which will occur when ToolBook finishes executing the current handler or `lockScreen` is set back to `false`.

EXAMPLES
```
-- reset position of all fields but lock the screen first
ls = lockScreen of viewer "help"
lockScreen of viewer "help" = true
step k from 1 to 10
    obj = field "fld" & k
    position of obj = resetPosition of obj
end
lockScreen of viewer "help" = ls
```

log()

Logarithmic Values

SYNTAX `log(<number>,<base>)`

DESCRIPTION Returns the logarithm of a number to the base you specify.

PARAMETERS

PARAMETER	DESCRIPTION
`<number>`	An expression that yields a positive number.
`<base>`	An expression that yields a positive number.

EXAMPLES
```
x = log(10)            -- equals 1
x = log(8, 2)          -- equals 3
x = log(86, 2.7182818)  -- equals 4.454347
```

lowercase()

String Functions

SYNTAX `lowercase(<expression>)`

DESCRIPTION Converts all of the characters in the expression to lowercase.

Characters that lack a lowercase equivalent in the `ANSI` character set remain unchanged. This includes characters such as numbers.

RETURNS Returns a string value containing the modified expression.

NOTES A character that is already lowercase will remain lowercase.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that results in a string value.

EXAMPLES

```
-- Capitalize each word in a string, ensuring all other characters
-- are lowercase
to handle buttonClick
    txt = lowercase(text of field "poem")
    step k from 1 to wordCount(txt)
        curWord = word k of txt
        uc = upperCase(character 1 of curWord)
        character 1 of curWord = uc
        word k of txt = curWord
    end
    text of field "poem" = txt
end
```

magenta

Color Constants

DESCRIPTION Represents the color `magenta`. This is equivalent to the `RGB` value of `255,0,255` and the `HLS` value of `300,50,100`.

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES

```
rgbFill of field "list" = magenta
```

```
if rgbStroke of rectangle "drop area" = magenta
    rgbStroke of rectangle "drop area" = 0,191,0
end
```

```
strokeColor of ellipse id 14 = magenta
```

magnification

Property

DESCRIPTION A property of a viewer that specifies the magnification of the contents of the viewer.

NOTES You can get or set this property.

To change the magnification of a page displayed in a viewer you can also use the `magnify` command.

VALUES `1`, `2`, `4`, `8`, or `16`, indicating the power of magnification in the viewer.

EXAMPLES

```
magnification of mainWindow = 4
```

mainWindow

System Property

DESCRIPTION A system property that specifies the viewer used as the ToolBook Main window.

NOTES Use this property to control the Main window from any viewer or book script. Using `mainWindow` is effectively a shortcut to saying `viewer id 0 of this book`

VALUES A reference to `viewer id 0` of the book viewed in the Main window.

EXAMPLES

```
-- standard sharedScript code to close current viewer
to handle buttonClick
    if this window <> mainWindow
        close this window
    end
end
```

majorVersionNumber

Property

DESCRIPTION A book property that specifies a list containing the first three [of four] numbers of the book's file version.

NOTES This is a read only property and cannot be set. The value of this property is determined by the version of ToolBook used to create the book.

VALUES A list of three positive numbers.

EXAMPLES

```
ver = majorVersionNumber of this book
```

make

Notification Message

DESCRIPTION Sent to an object just after it is created.

If you create a `group` by pasting it in a book, the make message is sent only to the `group`, not to the objects in it.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
-- Sets color of a new button to standard color for the page
to handle make
    if object of target is "button"
        fillColor of target = pFill of this page
        strokeColor of target = pStroke of this page
    end
    forward
end
```

matColor

Property

DESCRIPTION A property of a viewer that specifies the color used by ToolBook to fill any exposed sections of the viewer's client area.

NOTES You can get or set this property. The `matColor` is visible in areas of the viewer that are not covered by the client area. This is typically seen when you maximize your application.

VALUES A list of three `HLS` values, or a valid ToolBook `color constant`. The default is `0,50,0`.

EXAMPLES

```
matColor of viewer "help" = red
```

max()

Statistical Values

SYNTAX `max(<list>)`

DESCRIPTION Returns the highest value in a list of numbers.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<list>	An expression that results in a list of numbers.

EXAMPLES

```
lst = max(14,35,668,85,94)  -- results in 668
```

```
-- determine maximum character length of all page names
x = null
step k from 1 to pageCount of this book
    push charCount(name of page k) onto x
end
answ = max(x)
```

maximumScore

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of a question object which specifies the maximum possible score for the question.

NOTES You can get but not set this property in OpenScript as well as in the Actions Editor.

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

EXAMPLES

```
text of field "ScoreInfo_max" = maximumScore of group "Multiple Choice"
```

maximumSize

Property

DESCRIPTION A property of a viewer that specifies the viewer's maximum allowable size in `pixels`.

NOTES You can get or set this property.

A user cannot resize a viewer to be any larger than the size specified by this property. OpenScript ignores this property when you set the bounds of a viewer in a script.

VALUES `None`, or a list of two whole numbers in `pixels` that specify the height and width of a viewer's bounding rectangle.

If `none`, the maximum size of the window is limited only by the size of the screen.

EXAMPLES

```
maximumSize of viewer "help" = 400,600
```

mediaBounds

Property

DESCRIPTION A stage property that specifies the location of the stage's media display area in `pixels`.

The location is relative to the upper-left corner of the display area's client window.

NOTES This is a read only property and cannot be set.

VALUES A list of four numbers in `pixels`.

The first two numbers specify the `x,y` coordinates of the upper-left corner of the client window. The last two numbers specify the `x,y` coordinates of the lower-right corner.

EXAMPLES

```
bnds = mediaBounds of stage "player"
```

mediaOpen

Property

DESCRIPTION A stage property that specifies the media reference for a clip or bitmap resource displayed in the stage.

NOTES This is a read only property and cannot be set.

VALUES A reference to a clip or bitmap that is currently displayed in the stage. This value will be `null` if no clip or bitmap is displayed.

EXAMPLES

```
if mediaOpen of stage "player" <> null
    request "Sorry, you must complete this task before exiting."
end
```

mediaPlaying

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of a media player object that specifies the current position of the media.

NOTES You can get but not set this property in OpenScript as well as in the Actions Editor.

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

VALUES `True` if the media is still playing, otherwise `false`.

EXAMPLES

```
if mediaPlaying of group "UMP-1" = true
    show field "msg12"
end
```

mediaPosition

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of a media player object that specifies the current position of the media.

NOTES You can get but not set this property in OpenScript as well as in the Actions Editor.

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

VALUES The current position of the media, or `0` if the media is stopped.

EXAMPLES

```
text of field "currentPosition" = mediaPosition of group "UMP-1"
```

mediaSize

Property

DESCRIPTION A stage property that specifies the media reference for a clip or bitmap resource displayed in the stage.

NOTES This is a read only property and cannot be set.

VALUES A reference to a clip or bitmap that is currently displayed in the stage.

This value will be `null` if no clip or bitmap is displayed.

EXAMPLES

```
if mediaOpen of stage "player" <> null
    -- user still viewing clip, not allowed to exit at this time
    request "Sorry, you must complete this task before exiting."
end
```

menuBar

Resource Type

DESCRIPTION The resource type name for a `menu bar` resource.

Menu bar resources are referenced in OpenScript by `name` or by `ID` using the `menuBar` keyword. The `ID` is determined by the ToolBook system, but you can assign any name to the resource.

You can set a `menu bar` resource for the `menuBar` property of any viewer.

When you set the Main window's `menuBar` property to a menu bar resource, it becomes the `Reader`-level menu bar.

EXAMPLES

```
menuBar of viewer "ToolMaker" = menuBar "tools"
menuBar of mainWindow = menuBar "customized"
```

menuBar

Property

DESCRIPTION A property of a viewer that specifies the menu bar resource assigned to the viewer.

NOTES You can get or set this property.

This property can be set for the `Reader` level menu bar of the ToolBook Main window and for popup windows.

Child windows cannot have a menu bar. This means that Neuron cannot have a menu bar since `Neuron` is displayed as a child window of the browser window.

VALUES A reference to a menu bar resource, or `null`.

If `null`, no menu bar is assigned to the viewer.

The default for all viewers except the Main window is `null`.

The default setting for the Main window (`viewer id 0`) is `menuBar ID 100`.

The menu bar resource assigned to the `menuBar` property of the Main window becomes the book's `Reader`-level menu bar.

EXAMPLES

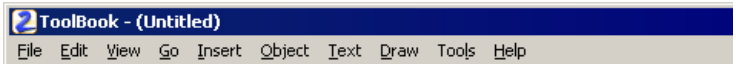
```
menuBar of viewer "help" = null
menuBar of viewer "dialog" = menubar "form"
```

menuBar

System Object

DESCRIPTION The object type name for a `menu bar` in a viewer.

NOTES You can show or hide a menu bar in the target window using the `show menuBar` or `hide menuBar` statement.



When a menu bar is hidden, its accelerator keys (such as `F3` or `Ctrl+S`) still work. To disable the accelerator keys as well, you must remove the menu item.

menuEnabled()

Menu Command/Function

SYNTAX

```
menuEnabled(<name | alias> [in <menu reference> [in <menu reference>]
  ... ] [at <level>] [,<viewer reference>])
```

DESCRIPTION Determines if a menu is `enabled` or `disabled`.

RETURNS Returns `true` if the specified menu in the target window is `enabled`.

Returns `false` if the menu is `disabled`.

If the specified menu in the target window does not exist, this function returns `null`.

NOTES You can refer to a menu in a submenu by using the in `<menu reference>` parameter. You can use this parameter multiple times to refer to a cascading menu with multiple submenus.

PARAMETERS

PARAMETER	DESCRIPTION
`<name | alias>`	The `name` of the menu as it appears on the menu bar, or the `alias` assigned to the menu.
`<menu reference>`	A valid reference to an existing `menu` or `submenu` in the target window. If more than one menu shares the same reference, ToolBook acts on the first menu from the left on the menu bar.
`<level>`	`Author` or `Reader`; if a level is not specified, the default is the current working level when the function is executed.
`<viewer reference>`	A valid reference to the viewer that owns the menu bar resource with the menu. This parameter is optional.

EXAMPLES

```
to handle enterMenu
    if menuEnabled("align" in menu "controls") = true and \
        selection is null
        disable menu "align" in menu "controls"
    end
    forward
end
```

menuItemChecked()

Menu Command/Function

SYNTAX

```
menuItemChecked(<name | alias> [in <menu reference> [in <menu
  reference>] ... ] [at <level>] [,<viewer reference>])
```

DESCRIPTION

Determines if a menuItem is currently `checked`.

RETURNS

Returns `true` if the menu item on a menu in the target window is `checked`.

Returns `false` if the menu item is `unchecked`.

If the specified menu item is not on a menu in the target window, this function returns `null`.

NOTES

You can refer to a menu item in a submenu by using the in `<menu reference>` parameter. You can use this parameter multiple times to refer to a cascading menu with multiple submenus.

PARAMETERS

PARAMETER	DESCRIPTION
`<name | alias>`	The `name` of the menu item as it appears on the menu, or the `alias` assigned to the menu item.
`<menu reference>`	A valid reference to an existing `menu` or `submenu` in the target window. If more than one menu shares the same reference, ToolBook acts on the first menu from the left on the menu bar.
`<level>`	`Author` or `Reader` or `both`; if a level is not specified, the default is the current working level when the function is executed.
`<viewer reference>`	A valid reference to the viewer that owns the menu bar resource with the menu. This parameter is optional.

EXAMPLES

```
to handle menuItemSelected mName, mAlias
    conditions
        when mAlias = "showOptions"
            if menuItemChecked("showOptions" in menu "Window") = true
                uncheck menuItem "showOptions" in menu "Window"
                hide viewer "Options"
            else
                check menuItem "showOptions" in menu "Window"
                show viewer "Options"
            end
    else
        forward
    end
end
```

menuItemEnabled()

Menu Command/Function

SYNTAX

```
menuItemEnabled(<name | alias> [in <menu reference> [in <menu
  reference>] ... ] [at <level>] [,<viewer reference>])
```

DESCRIPTION

Determines if a menuItem is currently `enabled`.

RETURNS

Returns `true` if the menu item on a menu in the target window is `enabled`.

Returns `false` if the menu item is `disabled`.

If the specified menu item is not on a menu in the target window, this function returns `null`.

NOTES

You can refer to a menu item in a submenu by using the in `<menu reference>` parameter. You can use this parameter multiple times to refer to a cascading menu with multiple submenus.

PARAMETERS

PARAMETER	DESCRIPTION
<name | alias>	The name of the menu item as it appears on the menu, or the alias assigned to the menu item.
<menu reference>	A valid reference to an existing menu or submenu in the target window. If more than one menu shares the same reference, ToolBook acts on the first menu from the left on the menu bar.
<level>	Author or Reader or both; if a level is not specified, the default is the current working level when the function is executed.
<viewer reference>	A valid reference to the viewer that owns the menu bar resource with the menu. This parameter is optional.

EXAMPLES

```
to handle menuItemSelected mName, mAlias
    conditions
        when mName = "undo"
            if menuItemEnabled("undo" in menu "edit") = true
                enabled of button "undo" = true
            else
                enabled of button "undo" = false
            end
    else
        forward
    end
end
```

menuItemPosition()

Menu Command/Function

SYNTAX

```
menuItemPosition(<name | alias> [in <menu reference> [in <menu
  reference>] ... ] [at <level>] [,<viewer reference>])
```

DESCRIPTION

Determines the position of the specified menu item.

RETURNS

Returns a number representing the position of the menu item.

If the specified menu item is not on a menu in the target window, this function returns null.

NOTES

You can refer to a menu item in a submenu by using the in <menu reference> parameter. You can use this parameter multiple times to refer to a cascading menu with multiple submenus.

PARAMETERS

PARAMETER	DESCRIPTION
<name | alias>	The name of the menu item as it appears on the menu, or the alias assigned to the menu item.
<menu reference>	A valid reference to an existing menu or submenu in the target window. If more than one menu shares the same reference, ToolBook acts on the first menu from the left on the menu bar.
<level>	Author or Reader or both; if a level is not specified, the default is the current working level when the function is executed.
<viewer reference>	A valid reference to the viewer that owns the menu bar resource with the menu. This parameter is optional.

EXAMPLES

```
get menuItemPosition("Style" in menu "Options")
```

menuItemSelected

Notification Message

SYNTAX `menuItemSelected <name>,<alias>`

DESCRIPTION Sent when the user selects a `menu item`. The `menuItemSelected` message is sent to the `viewer` that owns the menu bar.

ToolBook's default response to this message, when it reaches the `system` level, is to send the specific menu event message for the selected menu item.

If ToolBook finds a handler for the `menuItemSelected`, ToolBook does not send a menu event message (the `alias` or `name`) for a menu item when it is chosen. ToolBook sends a menu item's `alias` or `name` only when the `menuItemSelected` message reaches the `system` level. The `menuItemSelected` message will reach the system level if you don't write a `handler` for it, or if you write a handler for the `menuItemSelected` message that contains a `forward to system` statement.

To be certain that a handler for the `menuItemSelected` message does not interfere with handlers for other menu event messages, you should `forward` the `menuItemSelected` message any time you do not provide specific support for a menu item in the `menuItemSelected` handler. Use `forward to system` instead of `forward` to prevent a system book from interfering with menu event messages.

You can use the `<name>` parameter to determine if the user pressed a keyboard accelerator to select a menu item. When the user presses an accelerator key to select a menu item, the `menuItemSelected` message is sent with `null` as the value for `<name>`.

PARAMETERS

PARAMETER	DESCRIPTION
`<name>`	The name of the selected menu item. A value of `null` indicates that the user has pressed an accelerator key to select the menu item.
`<alias>`	The alias of the selected menu item.

EXAMPLES

```
--This script handles menu item messages
to handle menuItemSelected mMenuItem, mAlias
    if mAlias contains "Chapter"
        go to page mAlias
    else
        forward
    end
end
```

menuPosition()

Menu Command/Function

SYNTAX

```
menuPosition(<name | alias> [in <menu reference> [in <menu reference>]
  ... ] [at <level>] [,<viewer reference>])
```

DESCRIPTION Determines the position of the specified menu.

RETURNS A number representing the menu position.

If the specified menu in the target window does not exist, this function returns `null`.

NOTES You can refer to a menu in a submenu by using the in `<menu reference>` parameter. You can use this parameter multiple times to refer to a cascading menu with multiple submenus.

PARAMETERS

PARAMETER	DESCRIPTION
<name | alias>	The name of the menu as it appears on the menu bar, or the alias assigned to the menu.
<menu reference>	A valid reference to an existing menu or submenu in the target window. If more than one menu shares the same reference, ToolBook acts on the first menu from the left on the menu bar.
<level>	Author or Reader or both; if a level is not specified, the default is the current working level when the function is executed.
<viewer reference>	A valid reference to the viewer that owns the menu bar resource with the menu. This parameter is optional.

EXAMPLES

```
pos = menuPosition("Edit")
```

methods

UNDOCUMENTED Property

DESCRIPTION Contains a list of methods available to a control.

WARNING THIS IS AN UNDOCUMENTED FEATURE. What that means is that you can use it but you will most likely be unable to get assistance from Click2learn on how to use it, or troubleshoot it.

NOTES You cannot set this property.

All method names, such as findValue, are referenced in ToolBook with and ext preceding the method name. For example extFindValue.

VALUES A comma separated list of method names, with ext preceding each method name.

EXAMPLES

```
request methods of tList "master list"
```

midString()

Nonexistent String Functions

SYNTAX

```
midString(<string>,<start pos>,<num chars>)
```

DESCRIPTION Extracts a number of characters from within a string.

RETURNS Returns <num chars> characters from within the string starting with character <start char>.

NOTES Although this function does not exist in ToolBook you can add it by putting this script in the script of your book at which point you can call this function from anywhere in that book.

```
to get midString str, startPos, numChars
    return chars startPos to (startPos + numChars - 1) of str
end
```

Also see leftString() and rightString().

PARAMETERS

PARAMETER	DESCRIPTION
<string>	An expression that results in a string value.
<start pos>	An expression that results in a whole number.
<num chars>	An expression that results in a whole number.

EXAMPLES

```
-- Characters 6 and 7 of a part number is the bin location.
-- Find the bin number for this part.
to handle buttonClick
    partNum = textline 44 of field "allParts"
    binNumber = midString(partNum,6,2)
    put binNumber into text of field "bin"
end
```

min()

Statistical Values

SYNTAX `min(<list>)`

DESCRIPTION Returns the lowest value in a list of numbers.

ACTIONS EDITOR The Actions Editor also support s this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<list>	An expression that results in a list of numbers.

EXAMPLES

```
lst = min(14,35,668,85,94)  -- results in 14
```

```
x = null
step k from 1 to pageCount of this book
    push charCount(name of page k) onto x
end
answ = min(x)
```

minimumScore

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of a question object which specifies the minimum possible score for the question.

NOTES You can get but not set this property in OpenScript as well as in the Actions Editor.

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

EXAMPLES

```
text of field "ScoreInfo_min" = minimumScore of group "Multiple Choice"
```

minimumSize

Property

DESCRIPTION A property of a viewer that specifies the viewer's minimum allowable size in `pixels`.

NOTES You can get or set this property.

A user cannot resize a viewer to be any smaller than the size specified by this property. OpenScript ignores this property when you set the bounds of a viewer in a script.

VALUES `None`, or a list of two whole numbers in `pixels` that specify the height and width of a viewer's bounding rectangle.

If `none`, the minimum size will be the smallest size allowed by Windows.

EXAMPLES

```
minimumSize of viewer "help" = 20,20
```

minorVersionNumber

Property

DESCRIPTION A book property that specifies the fourth number of the book's file version.

NOTES You can get or set this property. When a book is first created this property defaults to zero.

VALUES A positive whole number.

EXAMPLES

```
-- set the revision number of this book
minorVersionNumber of this book = 16
```

mmBackgroundPalette

Clip Property

DESCRIPTION A property of a clip that specifies whether the book's palette or a visual media clip's palette has the higher priority.

NOTES You can get or set this property.

This property applies only to visual media clips.

VALUES `True` or `false`.

The default is `true`.

If `true`, the book's palette has higher priority. ToolBook draws media in "background" mode and the clip has access only to palette entries that are not already in use.

If `false`, the visual media clip's palette has higher priority. ToolBook draws the visual media clip in "foreground" mode and can only set the palette entries that are not used by the visual media clip's palette. Setting `mmBackgroundPalette` to `false` will most likely cause palette shifts between pages, but ensures that a visual media clip will use all of its colors.

EXAMPLES

```
mmBackground of clip "waterfall" = true
```

mmBeginPoint

Clip Property

DESCRIPTION A property of a clip that specifies the beginning of the clip relative to the beginning of its media source.

NOTES You can get or set this property.

This property is not valid for still-image media types.

VALUES An offset position in a valid time format for the specified clip. The time format is determined by the value of the clip's `mmTimeFormat` property.

EXAMPLES

```
mmBeginPoint of clip "waterfall" = "00:00:1:00"
```

mmClipHandle

Clip Property

DESCRIPTION A property of a clip that specifies the handle of the ToolBook window that owns a clip's play window.

NOTES You can get but cannot set this property.

The value for `mmClipHandle` usually refers to a ToolBook window that owns the play window created by a device driver.

VALUES A number assigned by Windows when the clip's play window opens.

If the clip is not a visual media type, the value is `null`.

SYNTAX

```
mmClose <media reference> [wait] [notify <object reference>]
mmClose all
```

DESCRIPTION

Closes a `clip` or `bitmap resource`.

NOTES

You must use the `mmClose` command to close any clip or bitmap resource that you opened with the `mmOpen` command.

Use the `mmIsOpen` clip property to determine if a particular clip is open. Use the `mmStatus` property of a clip to determine when to close the clip based on its current state (`closed`, `playing`, `paused`, and so on).

If the `wait` parameter is used, script execution will not continue until the clip or resource is `closed`.

If the `wait` parameter is not used, script execution continues as soon as the `mmClose` command is sent to the media device.

If the `notify <object reference>` parameter is used, ToolBook sends the `mmNotify` message to the specified object after the clip or resource closes.

If you plan on playing a piece of media only once, you should close it immediately after it finishes playing to free system resources. If you plan on playing a piece of media more than once, you will most likely want to leave it open.

PARAMETERS

PARAMETER	DESCRIPTION
`<media reference>`	A reference to the `clip`, or `bitmap resource` to be closed. To close all clips in the application, use the syntax `mmClose all`.
`wait`	Specifies that the media device does not return control to the application until the media device, file, or resource is closed. This parameter is optional.
`<object reference>`	A reference to the object to be notified.

EXAMPLES

```
--Closes a clip and sends notification to the page
mmClose clip "demoVideo" notify this page

--Closes a bitmap resource
mmClose bitmap "forest"

--Closes all clips in the application
mmClose all
```


SYNTAX

```
mmCue <media reference> [wait] [notify <object reference>]
```

DESCRIPTION

Cues an open `clip` to play from its start point

NOTES

You must first open the clip with the `mmOpen` command.

This command prepares media to play immediately and is useful for playing clips from `videotape`, `videodisc`, `CD-ROM`, or other formats in which it can take a substantial amount of time to locate a particular position before playing.

If the `wait` parameter is used, script execution will not continue until the clip is `cued`.

If the `wait` parameter is not used, script execution continues as soon as the `mmCue` command is sent to the media device.

If the notify `<object reference>` parameter is used, ToolBook sends the `mmNotify` message to the specified object after the clip is cued.

The clip must be open for the `mmCue` command to affect it.

PARAMETERS

PARAMETER	DESCRIPTION
`<media reference>`	A reference to the `clip`, or `bitmap resource` to be cued.
`wait`	Specifies that the media device does not return control to the application until the `clip` or `bitmap resource` is cued. This parameter is optional.
`<object reference>`	A reference to the object to be notified.

EXAMPLES

```
to handle enterPage
    mmOpen clip "ocean scene" wait
    mmCue clip "ocean scene"
    forward
end
```

mmDeviceAlias

Clip Property

DESCRIPTION A property of a clip that specifies the alias used as a reference for the clip (ToolBook automatically assigns the alias).

NOTES This property cannot be set.

You can use `mmDeviceAlias` with the `callMCI()` or `imageCommand()` functions for low-level media access.

VALUES A string ToolBook assigns as the alias for the open clip.

mmDeviceHandle

Clip Property

DESCRIPTION A property of a clip that specifies the `16-bit` handle of the clip's play window.

NOTES This p roperty cannot be set.

The value for `mmDeviceHandle` refers to the clip's play window, created by the device driver.

VALUES A `16-bit` number assigned by Windows when the clip's play window opens. If the clip is not a visual media type, the value is `0`.

mmDeviceHandle32

Clip Property

DESCRIPTION A property of a clip that specifies the `32-bit` handle of the clip's play window.

NOTES This property cannot be set.

The value for `mmDeviceHandle32` refers to the clip's play window, created by the device driver.

VALUES A `32-bit` number assigned by Windows when the clip's play window opens. If the clip is not a visual media type, the value is `0`.

mmEndPoint

Clip Property

DESCRIPTION A property of a clip that specifies the end of a clip relative to the beginning of its media source.

NOTES You can get or set this property.

This property is not valid for still-image media types.

VALUES An offset position in a valid time format for the specified clip. The time format is determined by the value of the clip's `mmTimeFormat` property.

EXAMPLES

```
mmEndPoint of clip "waterfall" = "00:00:10:25"
```

mmHide

Clip Commands

SYNTAX

```
mmHide <media reference> [wait] [notify <object reference>]
```

DESCRIPTION Hides a `clip` or `bitmap resource` that is currently shown.

NOTES This command is only valid for visual media types. If you attempt to use the `mmHide` command on a non-visual media type such as a .WAV file, an error occurs.

If the `wait` parameter is used, script execution will not continue until the clip or bitmap is `hidden`.

If the `wait` parameter is not used, script execution continues as soon as the `mmHide` command is sent to the media device.

If the `notify <object reference>` parameter is used, ToolBook sends the `mmNotify` message to the specified object after the clip or bitmap resource is hidden.

PARAMETERS

PARAMETER	DESCRIPTION
`<media reference>`	A reference to the `clip`, or `bitmap resource` to be hidden.
`wait`	Specifies that the media device does not return control to the application until the `clip` or `bitmap resource` is hidden.
`<object reference>`	A reference to the object to be notified.

EXAMPLES

```
mmHide clip "ocean scene" notify background "playMe"
mmHide bitmap "forest"
```

mmIsOpen

Clip Property

DESCRIPTION A property of a clip that specifies whether the clip is open.

NOTES This property cannot be set.

VALUES `True` is the clip is open, otherwise `false`.

mmLength

Clip Property

DESCRIPTION A property of a clip that specifies its length.

NOTES This property cannot be set.

This property is not valid for still-image media types.

VALUES A length in a valid time format for the specified clip. The time format is determined by the value of the clip's `mmTimeFormat` property.

mmMediaType

Clip Property

DESCRIPTION A property of a clip that specifies the media type associated with the clip's media device or file.

NOTES This property cannot be set.

VALUES The name of the media type: `animation`, `bitmap`, `cdAudio`, `digitalVideo`, `overlay`, `sequencer`, `vcr`, `videodisc`, or `waveAudio`.

mmNotify

Notification Message

SYNTAX `mmNotify <media reference>,<command>,<result>`

DESCRIPTION Sent when a media command finishes executing.

The `mmNotify` message is sent to the object specified in the `notify <object reference>` parameter of the particular media command.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
`<media reference>`	A reference to the `clip` or `bitmap` resource that is affected by the command.
`<command>`	The `name` of the command that acts on the `clip` or `bitmap` resource.
`<result>`	A keyword that indicates the result of the command statement when it executes: `aborted`, `failure`, `successful`, or `superseded`.

EXAMPLES

```
to handle buttonClick
    mmOpen clip "ocean scene"
    mmPlay clip "ocean scene" notify self
end

to handle mmNotify pClip, pCommand, pResult
    if pResult is not "successful"
        request "An error has occurred."
    end
    mmClose pClip
end
```

mmOpen

Clip Commands

SYNTAX `mmOpen <media reference> [wait] [notify <object reference>]`

DESCRIPTION Opens a `clip` or `bitmap resource`, verifies that its reference is valid, and prepares it to be played.

NOTES You must first `open` a clip or bitmap resource before you can use any other multimedia commands on it (with the exception of `mmPlay`) or access its properties. Use the `mmOpen` command in an `enterPage` handler to prepare media before playing it.

If the `wait` parameter is used, script execution will not continue until the clip or bitmap resource is opened.

If the `wait` parameter is not used, script execution continues as soon as the `mmOpen` command is sent to the media device.

If the `notify <object reference>` parameter is used, ToolBook sends the `mmNotify` message to the specified object after the clip or bitmap resource opens.

PARAMETERS

PARAMETER	DESCRIPTION
`<media reference>`	A reference to the `clip`, or `bitmap resource` to be opened.
`wait`	Specifies that the media device does not return control to the application until the `clip` or `bitmap resource` is open. This parameter is optional.
`<object reference>`	A reference to the object to be notified.

EXAMPLES

```
mmOpen clip "ocean scene" wait notify button "Play"
mmOpen bitmap "forest"
```

mmPause

Clip Commands

SYNTAX

```
mmPause <media reference> [wait] [notify <object reference>]
mmPause all
```

DESCRIPTION

Pauses an open clip at its current position while maintaining control of the media device channel.

NOTES

If you execute the `mmPlay` command with the clip after `mmPause`, the clip plays from the current position.

The `mmPause` command is similar to the `mmStop` command, with two exceptions: `mmPause` leaves visual media displayed onscreen (`mmStop` does not), and `mmStop` releases control of the media device channel to allow other clips to play (`mmPause` does not).

If the `wait` parameter is used, script execution will not continue until the clip pauses.

If the `wait` parameter is not used, script execution continues as soon as the `mmPause` command is sent to the media device.

If the `notify <object reference>` parameter is used, ToolBook sends the `mmNotify` message to the specified object after the clip stops playing.

The clip must be open for the `mmPause` command to affect it.

PARAMETERS

PARAMETER	DESCRIPTION
`<media reference>`	A reference to the `clip` to be paused.
`wait`	Specifies that the media device does not return control to the application until the `clip` or `bitmap resource` pauses. This parameter is optional.
`<object reference>`	A reference to the object to be notified.

EXAMPLES

```
--Pauses a clip and sends notification to the page
mmPause clip "demoVideo" notify this page

--Pauses all clips in the application
mmPause all
```

SYNTAX

```
mmPlay <media reference> [from <start position>] [to <end position>]
  [in <stage reference>] [autoclose | hold | release] [wait]
  [notify <object reference>]
```

DESCRIPTION

Plays a clip

NOTES

If the clip is *not* already open, ToolBook automatically opens the clip before playing it and automatically closes it when the play operation completes. In this scenario the `[autoclose | hold | release]` settings are not valid options and will be ignored if specified.

If a clip is open and no `<start position>` is specified, the clip plays from its current position by default. If no `<end position>` is specified, the clip plays to its end.

If you attempt to play a clip that has reached its end, ToolBook automatically rewinds the clip to its beginning and plays from that point.

If the `wait` parameter is used, script execution will not continue until the clip finishes playing. If the wait parameter is not used, script execution continues as soon as the `mmPlay` command is sent to the media device.

If the `notify <object reference>` parameter is used, ToolBook sends the `mmNotify` message to the specified object after the clip finishes playing.

PARAMETERS

PARAMETER	DESCRIPTION
`<media reference>`	A reference to the `clip` or `bitmap` resource to be played.
`<start position>`	A `position` in a valid time format for the clip; start (plays from the beginning of the clip, regardless of the current position); or 0 (same as start). This parameter is optional.
`<end position>`	An offset `position` (from the beginning of the clip) in a valid time format. This parameter is optional.
`<stage reference>`	A reference to a ToolBook stage. This parameter is optional. If the `in <stage reference>` parameter is not included, the clip or bitmap will play in its own window.
`autoclose`	Specifies that media closes automatically when it finishes playing, or when it is stopped. This parameter is optional and is not valid when either the `hold` or `release` parameter is used.
`hold`	Specifies that media `pauses` when it finishes playing. When media is paused, it retains its access to the device channel. This parameter is optional and is not valid when either the `autoclose` or `release` parameter is used.
`release`	Specifies that media stops when it finishes playing. When media is stopped, it releases its access to the device channel. This parameter is optional and is not valid when either the `autoclose` or `hold` parameter is used.
`wait`	Specifies that the media device does not return control to the application until the clip or bitmap finishes playing. This parameter is optional.
`<object reference>`	A reference to the object to be notified.

EXAMPLES

```
mmPlay clip "ocean scene" in stage "demoView" autoclose
mmPlay clip "orchestra" from "00:02" to "00:14" wait
mmPlay clip "film1" in stage "movie" hold notify self
```

mmPlayable

Clip Property

DESCRIPTION A property of a clip that specifies whether the clip can be played on the current system.

NOTES This property cannot be set.

VALUES `True` or `false`.

`True` means the clip is playable, otherwise `false`.

mmPosition

Clip Property

DESCRIPTION A property of a clip that specifies the current position of the clip in relation to its start point (the value of `mmBeginPoint`).

NOTES This property cannot be set.

This p roperty is not valid for still-image media types.

VALUES A position in a valid time format for the specified clip. The time format is determined by the value of the clip's `mmTimeFormat` property.

mmPriority

Clip Property

DESCRIPTION A property of a clip that specifies the clip's priority relative to other clips that use the same media device channel.

NOTES You can get or set this property.

A clip with a lower priority will be stopped to allow a clip with a higher priority to play.

VALUES A whole number between `0` (lowest priority) and `1000` (highest priority).

If the channel is not available, the clip with the highest priority gets first access.

If two clips have the same setting for mmPriority, the clip that has access to the channel retains it.

A new clip must have higher priority than the current clip to gain access to the channel.

mmRewind

Clip Commands

SYNTAX

```
mmRewind <media reference> [wait] [notify <object reference>]
mmRewind all
```

DESCRIPTION Stops a clip if it is playing and sets the current position to its beginning.

NOTES The clip must be open for the `mmRewind` command to affect it.

If the `wait` parameter is used, script execution will not continue until the clip finishes rewinding.

If the `wait` parameter is not used, script execution continues as soon as the `mmRewind` command is sent to the media device.

If the `notify <object reference>` parameter is used, ToolBook sends the `mmNotify` message to the specified object after the clip rewinds.

PARAMETERS

PARAMETER	DESCRIPTION
<media reference>	A reference to the clip or media type to rewind.
wait	Specifies that the media device does not return control to the application until the clip rewinds. This parameter is optional.
<object reference>	A reference to the object to be notified.

EXAMPLES

```
mmRewind clip "demoVideo" notify this page
mmRewind all
```

mmSearchCD

Clip Property

DESCRIPTION A property of a clip that determines whether ToolBook searches for media in the CD-ROM directories specified for the CDMediaPath property.

NOTES You can get or set this property.

VALUES True or false.

The default is false.

If true, ToolBook searches for clips in the CD-ROM directories specified for the CDMediaPath property.

mmSearchHD

Clip Property

DESCRIPTION A property of a clip that determines whether ToolBook searches for media in the hard disk directories specified for the HDMediaPath property.

NOTES You can get or set this property.

VALUES True or false.

The default is false.

If true, ToolBook searches for clips in the hard disk directories specified for the HDMediaPath property.

mmSeek

Clip Commands

SYNTAX

```
mmSeek <media reference> to <position> [from end] [wait]
  [notify <object reference>]
```

DESCRIPTION Seeks to a specified position in an open clip.

NOTES By default, seek operations are offset from the beginning of the clip. If the from end parameter is used, ToolBook seeks backward from the end of the clip.

The clip must be open for the mmSeek command to affect it.

If the wait parameter is used, script execution will not continue until the seek operation is completed.

If the wait parameter is not used, script execution continues as soon as the mmSeek command is sent to the media device.

If the notify <object reference> parameter is used, ToolBook sends the mmNotify message to the specified object after the seek operation is complete.

PARAMETERS

PARAMETER	DESCRIPTION
<media reference>	A reference to the clip in which the seek operation occurs.
<position>	A position in a valid time format for the clip.
from end	Specifies that ToolBook seeks backward from the end of the clip. This parameter is optional.
wait	Specifies that the media device does not return control to the application until the seek operation is completed. This parameter is optional.
<object reference>	A reference to the object to be notified.

EXAMPLES

```
mmSeek clip "film1" to 10500 wait
mmSeek clip "film1" to 200 from end notify self
```

mmShow

Clip Commands

SYNTAX

```
mmShow <media reference> [in <stage reference>] [wait]
  [notify <object reference>]
```

DESCRIPTION

Shows a clip or bitmap resource.

NOTES

This command is only valid for visual media types.

For video and animation clips, the current frame is displayed (usually the first frame when a clip is first opened).

You must open a clip or bitmap using mmOpen before you can show it using mmShow. If you attempt to use the mmShow command on a non-visual media type such as wave audio, an error results.

If the wait parameter is used, script execution will not continue until the clip or bitmap is shown.

If the wait parameter is not used, script execution continues as soon as the mmShow command is sent to the media device.

If the notify <object reference> parameter is used, ToolBook sends the mmNotify message to the specified object after the clip or bitmap is shown.

PARAMETERS

PARAMETER	DESCRIPTION
<media reference>	A reference to the clip or bitmap resource to be shown.
<stage reference>	A reference to a ToolBook stage. This parameter is optional. If the in <stage reference> parameter is not included, the clip or bitmap displays in its own play window.
wait	Specifies that the media device does not return control to the application until the clip or bitmap is shown. This parameter is optional.
<object reference>	A reference to the object to be notified.

EXAMPLES

```
mmShow clip "film1" in stage "movie" notify self
mmShow bitmap "actor1" in stage "movie"
```

mmSource

Clip Property

DESCRIPTION A property of a clip that specifies the media reference associated with the clip's media source device or file.

NOTES You can get or set this property.

ToolBook locates media source files for clips using reference information stored with the clip's `mmSource` property, in conjunction with the values stored in the `CDMediaPath` and `HDMediaPath` book properties.

The media path values include a list of drives and subdirectories ToolBook searches to find media source files.

VALUES A reference to a media file or a media device (such as `cdAudio` or `videodisc`).

When you set a media search path and create a clip, only the subdirectory path and file name of the media source are stored as part of the clip (as the value for `mmSource`).

For example, if you set the media path and created a clip from the file information `c:\TB\video\show.avi`, ToolBook would store the information as described in the table below.

INFORMATION	STORED	EXAMPLE
Media search path	In a list in `CDMediaPath` or `HDMediaPath` property	`c:\TB`
Clip reference	As the value for the clip's `mmSource` property	`video\show.avi`

When you play the clip, ToolBook adds the media search path to the clip reference so that it can find the media source file:

```
Media search path + clip reference = c:\TB + \video\show.avi
```

If you create a clip without setting a media search path, the clip reference (the value of `mmSource`) includes the entire path for the source media file, including the drive letter, subdirectories, and the media source file, for example: `c:\TB\video\show.avi.`

mmSourceLength

Clip Property

DESCRIPTION A property of a clip that specifies the length of the clip's media source device or file.

NOTES This property cannot be set.

This property is not valid for still-image media types.

VALUES A length in a valid time format for the clip. The time format is determined by the value of the clip's `mmTimeFormat` property.

mmSourcePosition

Clip Property

DESCRIPTION A property of a clip that specifies the current position of the clip relative to its media source device or file.

NOTES This property cannot be set.

This property is not valid for still-image media types.

VALUES A position in a valid time format for the clip.

The time format is determined by the value of the clip's `mmTimeFormat` property.

mmSourceTrackCount

Clip Property

DESCRIPTION A property of a clip that specifies the number of tracks in the clip's media source device or file.

NOTES This property cannot be set.

This property is not valid for still-image media types.

VALUES A whole number that represents the number of tracks.

mmSourceTrackInfo

Clip Property

DESCRIPTION A property of a clip that specifies the beginning and length of a track in the clip's media source device or file.

NOTES This property cannot be set.

This property is not valid for still-image media types.

VALUES A list that contains one or more pairs of two items in the time format for the specified media.

The first item in a pair represents the beginning of the track; the second item represents the track length.

The time format is determined by the value of the clip's `mmTimeFormat` property.

mmStatus

Clip Property

DESCRIPTION A property of a clip that specifies its current status.

NOTES You cannot set this property.

VALUES One of the following values: `closed`, `paused`, `playing`, `seeking`, or `stopped`.

VALUE	DESCRIPTION
`closed`	Clip has not been opened with `mmOpen`.
`paused`	Clip was playing but has been stopped with `mmPause`.
`playing`	Clip is `playing`.
`seeking`	Clip is seeking a new position or `rewinding`.
`stopped`	Clip has not begun playing or was stopped with `mmStop`.

EXAMPLES

```
if mmStatus of currentClip = "seeking"
    request "One moment please..."
end
```

mmStep

SYNTAX

```
mmStep <media reference> [back] by <distance> [wait]
  [notify <object reference>]
```

DESCRIPTION

Seeks forward from the current position in a clip by a specified amount.

NOTES

The clip must be open for the `mmStep` command to affect it.

If the `back` parameter is used, ToolBook seeks backward from the current position.

If the `wait` parameter is used, script execution will not continue until the `seek` operation is completed.

If the `wait` parameter is not used, script execution continues as soon as the `mmStep` command is sent to the media device.

If the `notify <object reference>` parameter is used, ToolBook sends the `mmNotify` message to the specified object after the seek operation is complete.

PARAMETERS

PARAMETER	DESCRIPTION
`<media reference>`	A reference to the clip in which the seek operation occurs.
`back`	Specifies that ToolBook seeks backward from the current position. This parameter is optional.
`<distance>`	A length in a valid time format for the clip.
`wait`	Specifies that the media device does not return control to the application until the seek operation is completed. This parameter is optional.
`<object reference>`	A reference to the object to be notified.

EXAMPLES

```
mmStep clip "film1" by "2000" notify self
mmSeek clip "film1" back by "0:03:00"
```

mmStop

SYNTAX

```
mmStop <media reference> [wait] [notify <object reference>]
mmStop all
```

DESCRIPTION

Stops a clip or bitmap resource from playing and holds its current position.

NOTES

If you execute the `mmPlay` command with the clip or bitmap resource after `mmStop` (or `mmPause`), it plays from the current position.

The `mmStop` command is similar to the `mmPause` command, except that `mmPause` retains control of the media device channel.

If the `wait` parameter is used, script execution will not continue until the clip or bitmap resource stops playing.

If the `wait` parameter is not used, script execution continues as soon as the `mmStop` command is sent to the media device.

If the `notify <object reference>` parameter is used, ToolBook sends the `mmNotify` message to the specified object after the clip or bitmap resource stops playing.

PARAMETERS

PARAMETER	DESCRIPTION
`<media reference>`	A reference to the clip, bitmap resource, or media type to be stopped.
`wait`	Specifies that the media device does not return control to the application until the clip or bitmap resource stops playing. This parameter is optional.
`<object reference>`	A reference to the object to be notified.

EXAMPLES

```
mmStop clip "demoVideo" notify this page
mmStop bitmap "forest"
mmStop all
```

mmTimeFormat

Clip Property

DESCRIPTION A property of a clip that specifies its time format.

NOTES You can get or set this property.

Each media type supports certain time formats.

This property is not valid for still-image media types.

You can use the value of `mmTimeFormat` with the following clip properties: `mmBeginPoint`, `mmEndPoint`, `mmLength`, `mmPosition`, `mmSourceLength`, `mmSourcePosition`, `mmSourceTrackInfo`, and `mmTrackInfo`.

VALUES A valid time format for the media type.

Possible values include: `milliseconds`, `frames`, `HMS` (hours:minutes:seconds), `TMSF` (tracks:minutes:seconds:frames), `HMSF` (hours:minutes:seconds:frames), `MS` (minutes:seconds), `MSms` (minutes:seconds:milliseconds), `MSF` (minutes:seconds:frames).

EXAMPLES

```
mmTimeFormat of clip "music" = "MS"
```

mmTrackCount

Clip Property

DESCRIPTION A property of a clip that specifies the number of tracks in the clip.

NOTES This property cannot be set.

This property is not valid for still-image media types.

VALUES A whole number that represents the number of tracks.

mmTrackInfo

Clip Property

DESCRIPTION A property of a clip that specifies the beginning and length of a track in the clip.

NOTES This property cannot be set.

This property is not valid for still-image media types.

VALUES A list that contains one or more pairs of two items in the time format for the specified media.

The first item in a pair represents the beginning of the track; the second item represents the track length.

The time format for the source position is determined by the value of the clip's `mmTimeFormat` property.

mmVisible

Clip Property

DESCRIPTION A property of a clip that specifies if the clip is visible.

NOTES This property cannot be set.

This property is valid only for visual media types.

VALUES `True` or `false`.

If `visible` the value is `true`, otherwise `false`.

mmVisualSize

Clip Property

DESCRIPTION A property of a clip that specifies the default size of its visual media when shown.

NOTES This property cannot be set.

This property is valid only for visual media types.

VALUES A list of two positive whole numbers in `pixels` that represent width and height.

mmVolume

Clip Property

DESCRIPTION A property of a clip that specifies the volume level associated with the clip.

NOTES You can get or set this property.

This property is valid only for clips that contain sound.

The setting for a clip's volume level represents the clip's volume as a percentage of the system volume.

If `mmVolume` is set to full and the system volume level is set to `50` percent, the volume level the user hears will be `50` percent.

VALUES `Mute` (play silently), `normal` (play at normal level), `full` (play at loudest level possible), or a number from `0` to `1000` that represents a percentage of the normal volume.

mmYield

Clip Commands

SYNTAX `mmYield()`

DESCRIPTION Yields processing time to allow media drivers to refresh buffers and process messages.

NOTES Use the `mmYield` command when a script is running at the same time as media to prevent media from breaking up and to ensure properties that return the current state of media (such as `mmStatus`) are updated.

PARAMETERS No parameters.

EXAMPLES

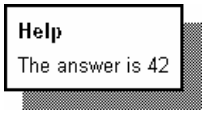
```
mmYield()
```

modalPopText()

TBWIN.DLL Dialog Box

SYNTAX `modalPopText(<title>,<text string>,<point>)`

DESCRIPTION Displays text in a read-only, popup window with a shadowed border. Script execution is halted until the popup window is dismissed.



RETURNS If successful, the function returns `0`.

Otherwise, the function returns this error:

`-1` Unable to open the window.

NOTES The window is sized appropriately for the text displayed and the position is adjusted if necessary to keep the window on the screen.

The text is displayed in the Arial font face.

The popup window is dismissed when the user presses a key or clicks the mouse.

As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT modalPopText(STRING,STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<title>`	A string that specifies the text to be displayed in bold above the text string in the popup window.
`<text string>`	A string that specifies the text to be displayed in the popup window.
`<point>`	A string containing a list of two numbers in `page units` that represent the upper-left corner of the popup window.

EXAMPLES

```
to handle buttonClick
    get modalPopText("Help","The answer is 42",sysmouseposition)
end
```

(modulus) mod

Arithmetic Operators

SYNTAX `<expression> mod <expression>`

DESCRIPTION Calculates the `modulus` by dividing the left expression by the right expression.

The integer (whole number) portion is discarded and only the remainder (partial number) is returned.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that results in a numeric value.

EXAMPLES

```
partialCarton = totalEggs mod 12

-- stores 1 in z
put 10 mod 3 into z
```

mouseEnter

Event Message

DESCRIPTION Sent to an object at `Reader` level when the `cursor` enters an object's bounds.

NOTE All objects on a `page` or `background` except `groups` receive the `mouseEnter` message automatically.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle mouseEnter
    oldCaption of self = caption of self
    caption of self = upperCase(caption of self)
    forward
end

to handle mouseLeave
    caption of self = oldCaption of self
    forward
end
```

mouseLeave

Event Message

DESCRIPTION Sent to an object at `Reader` level when the `cursor` leaves an object's bounds.

NOTE All objects on a `page` or `background` except `groups` receive the `mouseLeave` message automatically.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle mouseEnter
    oldCaption of self = caption of self
    caption of self = upperCase(caption of self)
    forward
end

to handle mouseLeave
    caption of self = oldCaption of self
    forward
end
```

mousePosition

Property

DESCRIPTION A property of a viewer that specifies the position of the `cursor`.

NOTES You can get or set this property.

VALUES A list of two whole numbers in `page units` specifying the location of the `cursor` on the page as `x,y` coordinates, measured from the upper-left corner of the client area of the viewer.

EXAMPLES

```
-- after user clicks on this button, move mouse pointer to 50,150
to handle buttonClick
    mousePosition of viewer "help" = 50,150
end
```

move

Object Command

SYNTAX

```
move [<object>] by <amount>
move [<object>] to <location>
```

DESCRIPTION

Moves one or more objects, a built-in ToolBook window or palette, or the selection either by a specified amount or to a specified location.

PARAMETERS

PARAMETER	DESCRIPTION
`<object>`	A valid reference to a movable object, a built-in ToolBook window or palette, or a list of objects. The default is the current selection.
`<amount>`	Specifies an offset from the object's current position as a list of two numbers, representing the horizontal offset from left to right and the vertical offset from top to bottom, respectively. For viewers and ToolBook's built-in windows and palettes, values are specified in `pixels`. For all other objects, values are specified in `page units`.
`<location>`	A list of two numbers that represents a point on the screen (for windows and palettes) or on the page (for objects). For viewers and ToolBook's built-in windows and palettes, values are specified in `pixels`. For all other objects, values are specified in `page units`.

EXAMPLES

```
move button "Next" by 1440,-1440
```

moved

Notification Message

DESCRIPTION

Sent at `Author` level to an object when its `position` property is changed.

If a multiple object selection is moved, all the objects receive the `moved` message.

If the object is a group, the group receives this message, but individual objects in the group **do not**.

If the object is a viewer, the `moved` message is sent at both `Reader` and `Author` levels. The moved message is sent when the viewer is `maximized` or returned to its `normal` size, but not when the viewer is `minimized` or when the `minimized` viewer's icon is moved.

If the `Undo` menu command is chosen after an object is moved, the `moved` message is not sent again to the object.

NOTE

As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
--Prevents an object from being moved too far to the left
to handle moved
    system svLimit1
    get item 1 of bounds of target
    if IT < svLimit1
        move target by (svLimit1 - It),0
    end
    forward
end
```

moveFile()

TBDOS.DLL File Function (General)

SYNTAX `moveFile(<file spec>,<destination>)`

DESCRIPTION Moves and optionally renames the specified file.

You cannot move a file from one drive to another with `moveFile()`. You must instead use `copyFile()` and then `removeFile()`.

RETURNS

1	Function was successful.
-2	Specified file was not found.
-3	Specified path was invalid.
-5	Access to the file was denied.
-8	Source file could not be opened.
-17	Specified paths for the source and destination files refer to different disk drives.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    INT moveFile(STRING,STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `moveFile32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<file spec>`	The name of the file to be moved or renamed.
`<destination>`	If a path is specified, the file is moved to that directory. If a file name is supplied, the file is renamed to that name. Otherwise, the file retains its original name.

EXAMPLES

```
-- Rename the lesson file
to handle buttonClick
    oldName = "c:\project\information.doc"
    newName = "c:\project\information.bak"
    get moveFile(oldName,newName)
end
```

moveFile32()

TBFILE32.DLL File Function (General)

SYNTAX `moveFile32(<file spec>,<destination>)`

DESCRIPTION Moves and optionally `renames` the specified file.

You cannot move a file from one drive to another with `moveFile32()`. You must instead use `copyFile32()` and then `removeFile32()`.

RETURNS `1` Function was successful.

Otherwise, the function returns one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    INT moveFile32(STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<file spec>	The name of the file to be moved or renamed.
<destination>	If a path is specified, the file is moved to that directory. If a file name is supplied, the file is renamed to that name. Otherwise, the file retains its original name.

EXAMPLES

```
-- Rename the lesson file
to handle buttonClick
    oldName = "c:\project\information.doc"
    newName = "c:\project\information.bak"
    get moveFile32(oldName,newName)
end
```

moveObject()

Object Function

SYNTAX

```
moveObject (<object reference>,<destination reference>)
```

DESCRIPTION

Moves a ToolBook object from one location to another without requiring you to navigate to the source location or use the `Clipboard`.

The move is performed by copying the object to the new location and, if that is successful, deleting the original object.

Note that the new object will have a different id than the original one had.

RETURNS

A reference to the moved object at the specified location, or `NULL` if the operation is unsuccessful.

PARAMETERS

PARAMETER	DESCRIPTION
<object reference>	A valid object or a page. If an object other than a page is specified, the second parameter can be a `page` or a `group` object on a `page` or `background`. If a `page` is specified, the second parameter must be another `page` or a `background`.
<destination reference>	A `page` or `group` if the first parameter is an object other than a `page`. If the first parameter is a `page`, this parameter must be a `page` or `background`.

EXAMPLES

```
--moves the button "chapter 1" to page "all chapters"
get moveObject(button "chapter 1", page "all chapters")

-- moves the current page to the page "engine model"
get moveObject(this page, page "engine model")
```

name

Property

DESCRIPTION

A property of all ToolBook objects that specifies a string (`name`) that refers to the object.

NOTES

Although you can get the `name` of a book, you cannot set the `name` property of a book.

For books, the `name` property specifies the file name of a book.

To set the `name` of a book, perform a `Save As` and specify a filename.

For all other objects, you can both get and set the `name` property.

VALUES For all objects except books, the value is a string of up to 32 characters.

By default an object has no name - or a value of `null`.

For books, the value is the valid path of the file.

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES
```
-- What drive letter is the book on?
driveLetter = chars 1 to 3 of name of this book
```

```
-- Script at Book Level
to handle buttonClick
    if name of target = "stop" AND object of target = "button"
        send exit
    end
    forward
end
```

new sharedScript

Resource Commands

SYNTAX
```
new sharedScript
```

DESCRIPTION Creates a new, blank `sharedScript` resource.

An object reference to the new sharedScript resource is placed in the variable IT.

EXAMPLES
```
new sharedScript
name of it = "dataValidation"
```

new viewer

Viewer Command

SYNTAX
```
new viewer
```

DESCRIPTION Creates a new `viewer` in the context of the target window.

NOTES The `new viewer` command returns a reference to the new viewer in the variable IT.

EXAMPLES
```
new viewer
name of IT = "Help"
```

next

Navigation Message

DESCRIPTION Sent to the page when `Next Page` is chosen from the `Go` menu.

You can also `send` this message using the `send next` statement. ToolBook's default response is to display the next page in the current book.

If `next` is sent when the user is on the last page of a book, ToolBook displays the first page of the book.

NOTE: Please see `skipNavigation` to see how this page property can affect your attempts to navigate.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES
```
to handle buttonClick
    send next
end
```

```
to handle next
    request "Sorry, you are not permitted to leave this page yet."
end
```

ninth

Ordinal Numbers

DESCRIPTION A value used to indicate the ninth item in a sequence or list. Also used to indicate relative position of pages or backgrounds.

EXAMPLES

```
-- The following paired examples show how you can use ordinal
-- references to make OpenScript easier to read. As you can see it is
-- possible to write your scripts with or without ordinal references.
go to the ninth page of this book
go to page 9 of this book

if ninth character of str = "X"
if character 9 of str = "X"

ninth word of text of field "lesson" = "Hello"
word 9 of text of field "lesson" = "Hello"

put "Cancelled:" into ninth character of name of button id 16
put "Cancelled:" into character 9 of name of button id 16
```

noDropImage

Property

DESCRIPTION A property that specifies the image that is displayed as the cursor when a source object is dragged over an object that does not accept drops during a `drag-and-drop` operation.

NOTES You can get or set this property.

The `dragImage` property specifies the cursor that is displayed when the user drags the cursor over an object that accepts a drop.

The `noDropImage` property specifies the cursor that is displayed when it is dragged over an object that does not accept drops.

VALUES A valid reference to a `bitmap`, `cursor resource`, or `icon`, or `null`; the default is `null`.

If `null`, ToolBook displays the `no-drop` system cursor.

EXAMPLES

```
noDropImage of ellipse "b3" = bitmap "star"
```

normalGraphic

Property

DESCRIPTION A button property that specifies a graphic resource assigned to be the resource shown when the button is not in a checked, disabled or inverted state.

NOTES You can get or set this property.

VALUES A valid resource reference.

You can assign an `icon`, `cursor`, or `bitmap resource`.

If `null`, ToolBook displays no graphic on the button.

EXAMPLES

```
normalGraphic of button "Print" = icon "PrinterReady"
```

not

Logic Operators

SYNTAX

```
not <expression>
```

DESCRIPTION Returns `true` if the expression evaluates to `false`. Otherwise returns `false`.

NOTES This is an odd operator but powerful, the Examples for how to use this.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<expression>	Any Expression.

EXAMPLES

```
-- Both of these IF statements below do the same thing.
-- This one uses the NOT operator
if not checked of button "apple"
    request "Don't you like apples"
end

-- This one doesn't use the NOT operator
if checked of button "apple" = false
    request "Don't you like apples"
end
```

notifyAfter

Handler Structure

SYNTAX `notifyAfter <message>`

DESCRIPTION Permits an object on a `Page` (or `Background`) to be notified when a page level event occurs.

The `notifyAfter` handler sends a message to the object immediately `after` the page receives the message.

NOTES Since messages in ToolBook always travel `UP` the hierarchy, object on a page don't get notified when page level events occur, such as `enterPage`. If you would like your page object to be notified when a page level event occurs you can use the `notifyAfter` (or `notifyBefore`) handler to request from ToolBook that your object also be notified.

When using the `notifyAfter` handler, your object will be sent the message `after` the page processes the message.

PARAMETERS

PARAMETER	DESCRIPTION
<message>	The name of any `message` that is normally received by a page.

EXAMPLES

```
-- clear the text of this field each time the page is entered
notifyAfter enterPage
    text of self = null
end
```

notifyAfterMessages

Property

DESCRIPTION A property of ToolBook objects which keeps track of which `notifyAfter` handlers are present in the object.

NOTES This is a read only property and cannot be set.

The book object, background, pages, and viewers do not have this `notifyAfterMessages` property as it is not permitted to put such handlers in these objects.

This is not a property that you would typically modify or adjust from a programming perspective. It is however a property that assists ToolBook in processing messages.

Anytime you write a `notifyAfter` handler in an object, the handler name is added to this property.

VALUES A comma separated list of handler names.

notifyBefore

Handler Structure

SYNTAX `notifyBefore <message>`

DESCRIPTION Permits an object on a `Page` (or `Background`) to be notified when a page level event occurs.

The `notifyBefore` handler sends a message to the object immediately `before` the page receives the message.

NOTES Since messages in ToolBook always travel `UP` the hierarchy, object on a page don't get notified when page level events occur, such as `enterPage`. If you would like your page object to be notified when a page level event occurs you can use the `notifyBefore` (or `notifyAfter`) handler to request from ToolBook that your object also be notified.

When using the `notifyBefore` handler, your object will be sent the message `before` the page processes the message.

PARAMETERS

PARAMETER	DESCRIPTION
`<message>`	The name of any `message` that is normally received by a page.

EXAMPLES

```
-- clear the text of this field each time the page is entered
notifyBefore enterPage
    text of self = null
end
```

notifyBeforeMessages

Property

DESCRIPTION A property of ToolBook objects which keeps track of which `notifyBefore` handlers are present in the object.

NOTES This is a read only property and cannot be set.

The book object, background, pages, and viewers do not have this `notifyBeforeMessages` property as it is not permitted to put such handlers in these objects.

This is not a property that you would typically modify or adjust from a programming perspective. It is however a property that assists ToolBook in processing messages.

Anytime you write a `notifyBefore` handler in an object, the handler name is added to this property.

VALUES A comma separated list of handler names.

notifyObjects

Property

DESCRIPTION A page property that indicates which objects on the page have a `notifyBefore` or `notifyAfter` handler in them.

A background property that indicates which objects on the background have a `notifyBefore` or `notifyAfter` handler in them.

NOTES You cannot set this property.

This property is not terribly useful from an OpenScript perspective. The property exists to assist ToolBook in properly handling the processing of `notifyBefore` and `notifyAfter` handlers.

VALUES Contains a comma separated list of object reference.

EXAMPLES

```
x = notifyObjects of this page
```

nper()

Financial Values

SYNTAX `nper(<rate>,<payment>,<present value>,<future value>[,<type>])`

DESCRIPTION A financial function that returns the number of periods required for an investment to reach a future value. This function is based on periodic, constant payments and a constant interest rate. Enter payments out of pocket as negative values, and enter income as positive values.

PARAMETERS

PARAMETER	DESCRIPTION
`<rate>`	A number representing the interest rate per period.
`<payment>`	A number representing the payment made each period; it must remain constant over the life of an investment.
`<present value>`	A number representing the present value, or the lump sum amount, that a series of future payments is worth at the present time.
`<future value>`	A number representing the future value, or the cash balance you want to attain after the last payment is made. If `<future value>` is omitted, it is assumed to be `0`.
`<type>`	Optional Parameter. A value of `true` or `false` that indicates when payments are due. `True` indicates that payments are due at the beginning of the period (annuity due), `false` at the end (ordinary annuity). If `<type>` is omitted, it is assumed to be `false`.

EXAMPLES

```
-- Calculates the number of payments required to reach a future value
x = nper(0.095/12,-500,-2500,35000,true)
```

npv()

Financial Values

SYNTAX `npv(<rate>,<values>[,<type>])`

DESCRIPTION A financial function that returns the present value of an investment based on a series of cash flow amounts at a given interest rate. The accuracy of this function depends on the accuracy of the cash flow values.

PARAMETERS

PARAMETER	DESCRIPTION
`<rate>`	A number representing the interest rate.
`<values>`	A list of numbers that must contain at least one positive number (income) and one negative number (payment out of pocket). The order of this list indicates the sequence in which the cash flows occur. If a non-numeric value is found in the list, ToolBook displays an `Execution Suspended` message.
`<type>`	A value of true or false that indicates when payments are due. `True` indicates that payments are due at the beginning of the period (annuity due), `false` at the end (ordinary annuity). If `<type>` is omitted, it is assumed to be `false`.

EXAMPLES

```
-- Determines how much an investment is currently worth
x = npv(0.095,"8356,8900,9844,10000,12560")
```

null

Special Constants

DESCRIPTION Represents an empty string. This is equivalent to "".

EXAMPLES

```
x = null
```

```
if text of field "list" = null
    request "Sorry, nothing to process at this time"
end
```

object

Property

DESCRIPTION A property of all ToolBook objects that specifies the `type` of object.

NOTES This is a read only property and cannot be set.

VALUES The object property of ToolBook objects will contain one of the following strings:

```
AngledLine, Arc, Background, Book, Button, ComboBox, Curve, Ellipse, Field,
Group, Hotword, IrregularPolygon, Line, OLE, Page, PaintObject, PictureObject,
Pie, Polygon, RecordField, Rectangle, roundedRectangle, Stage, Viewer
```

EXAMPLES

```
-- Script at Book Level
to handle buttonClick
    if object of target = "button"
        beep 3
    end
    forward
end
```

objectContainer()

Object Function

SYNTAX `objectContainer(<Object>,<ContainerList>)`

DESCRIPTION Finds a container for the given object from the input list of container types.

RETURNS The type of object for the parent of `<Object>` is inspected to see if it is included in the list of container types.

If so, then that object is returned.

If not, the type of the parent of the parent of `<Object>` is inspected, and so on up to the book itself.

If no parent object is found in the list of containers, then `null` is returned.

PARAMETERS

PARAMETER	DESCRIPTION
`<Object>`	Returns the object for a specified container.
`<ContainerList>`	A list of one or more container types for which to search. Container objects in ToolBook are `Group`, `Page`, `Background` and `Book`.

EXAMPLES

```
to handle ButtonClick
    bookRef = objectContiner(self,"book")
    increment clickCount of bookRef
end
```

objectCount

Property

DESCRIPTION A page property that specifies the number of object on a page.

A background property that specifies the number of objects on a background.

NOTES You cannot set this property.

The `objectCount` property counts the absolute number of object, whereas using this OpenScript `itemCount(objects of this page)` only counts the first level objects. First level objects don't include objects in groups or hotwords in fields.

VALUES A positive whole number representing the number of objects.

EXAMPLES

```
ct = objectCount of page "lesson"
```

```
qty = objectCount of background "course"
```

objects

Property

DESCRIPTION A `background` property that specifies the first level objects on a background.

A `page` property the specifies the first level objects on a page.

A `field` property which specifies the hotwords in the field.

A `recordfield` property which specifies the hotwords in the recordfield.

A `group` property that specifies the number of first level objects in a group.

NOTES You cannot set this property.

First level objects don't include objects in groups or hotwords in fields.

The objects property of a page does not include the `uniqueName` of each recordfield because recordfields exist on the background.

If the objects are grouped, only the `uniqueName` of the group appears in the list of objects for the group's parent.

Since the `objects` property of a page or background does not reveal all of the objects inside of groups which may be nested inside other groups, you may want to use the `getObjectList()` function instead.

VALUES A comma separated list containing the `uniqueName` of each object in the container object.

EXAMPLES

```
-- are there any hotwords in this field
ct = itemCount(objects of self)
if ct > 0
    -- yes there does seem to be hotwords in this field
    request "There are " & ct & " hotwords in this field."
end
```

objectType()

Object Function

SYNTAX `objectType(<object reference>)`

DESCRIPTION Returns the type of an object.

RETURNS The type of the object, including one of the following:

```
ToolBook          Internal ToolBook object
OCX               ActiveX object
VBX               Visual Basic Extension object
ActiveX16         OCX 16 bit control
ActiveX32         OCX 32 bit control
OleServer16       OLE Server 16 bit
OleServer32       OLE Server 32 bit
```

PARAMETERS

PARAMETER	DESCRIPTION
`<object reference>`	A valid reference to an object.

EXAMPLES

```
to handle buttonClick
    -- find any non ToolBook objects on this page
    objs = objects of this page
    step k from 1 to itemCount(objs)
        curObj = item k of objs
        if "Toolbook" is not in objectType(curObj)
            request curObj & " is not a ToolBook object."
        end
    end
end
```

of

Articles and Prepositions

DESCRIPTION A term used when expressing the ownership of an object or property. Typically seen when specifying the hierarchical relationship of an object.

EXAMPLES

```
x = text of field "trees" of page "list" of background "chapter 1"

myMadeUpProperty of field "trees" = "44993"
```

offset()

String Functions

SYNTAX

```
offset(<string>,<source>)
```

DESCRIPTION Locates the first occurrence of `<string>` in `<source>`.

RETURNS Returns the character position in which `<string>` was found in `<source>`. If `<string>` cannot be found within `<source>` then `0` is returned.

NOTES You will find in ToolBook that there are various ways to do the same thing. This means that you will encounter various coding styles as you look at other programmers code. For example here are three ways to see if a string exists within another string:

```
if offset(string1,string2) > 0
    request "Yes, match found."
end

if string1 is in string2
    request "Yes, match found."
end

if string2 contains string1
    request "Yes, match found."
end

-- Ok, and one more for the really insane
if not string1 is not in string2
    request "Yes, match found."
end
```

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<string>`	An expression that results in a string value.
`<source>`	An expression that results in a string value.

EXAMPLES

```
to handle buttonClick
    -- Correct bad backslashes in a URL
    x = text of field "URL"
    while "\" is in x
        pos = offset("\",x)
        char pos of x = "/"
    end
    text of field "URL" = x
end
```

onBackground

Property

DESCRIPTION A property of a viewer that specifies whether the page or its background is displayed in the viewer.

NOTES You can get or set this property.

Use the onBackground property to switch between a page and its background and make changes to the page or background in a viewer.

VALUES True or false.

If true, the background is active in the viewer, and the page is not visible.

If false, the page is active.

Setting onBackground to true is equivalent to sending the background message.

EXAMPLES

```
-- draw a new ellipse on the background
onBackground of this window = true
draw ellipse from 50,50 to 100,75
onBackground of this window = false
```

onto

Articles and Prepositions

DESCRIPTION A term used when manipulating stack data with the push command.

EXAMPLES

```
lst = objects of this page
while lst <> null
    pop lst into tmp
    if object of tmp = "button"
        push tmp onto listOfButtons
    end
end
```

open

Viewer Command

SYNTAX open <viewer reference>

DESCRIPTION Opens a viewer without showing it.

Use the open command to modify the runtime properties of the viewer before the viewer becomes visible. A viewer must be opened before any of its runtime properties can be set.

The open command has no effect on the ToolBook Main window.

NOTES ToolBook automatically opens a viewer if you show it using the show command.

PARAMETERS

PARAMETER	DESCRIPTION
<viewer reference>	A valid reference to a viewer or a list of viewers.

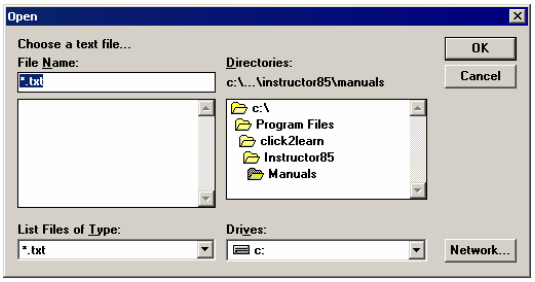
EXAMPLES

```
-- Opens a viewer before showing it
open viewer "customToolbar"
parentHandle of viewer "customToolbar" = windowHandle of viewer "layout"
show viewer "customToolbar"
```

SYNTAX `openDlg(<dir>,<ext>,<prompt>,<caption>)`

DESCRIPTION Displays a standard Windows Open dialog box.

This function will accept a long file name as input, but will return short file names only. Use `openDlgLFN()` if you need to return long file names.



RETURNS If no error occurs, `openDlg()` returns a string that contains the file name and path of the file the user specified.

Otherwise, the function returns `null` and sets `sysError` to one of these values:

`-1` User canceled the dialog box.
`-3` Memory allocation error.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING openDlg(STRING,STRING,STRING,STRING)
end
```

PARAMETERS

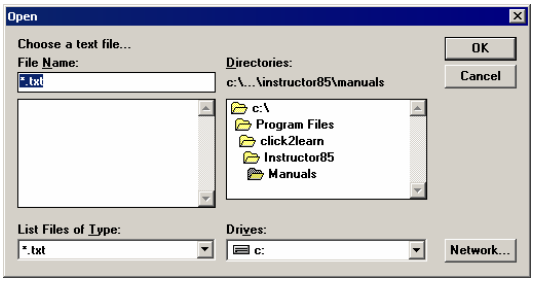
PARAMETER	DESCRIPTION
`<dir>`	The directory to be displayed. Use `"."` to specify the current directory, or `".."` to specify its parent directory, or include the complete path for the desired directory.
`<ext>`	The extension for the file names to be displayed. For example, `"*.tbk"` displays ToolBook files.
`<prompt>`	The text for the static control at the top of the dialog box, which usually provides instructions to the user.
`<caption>`	The caption for the title bar of the dialog box.

EXAMPLES

```
to handle openText
    get openDlg(".","*.txt","Choose a text file...","Open")
    if sysError is - 1
        break to system
    end if
    currentFile = upperCase(It)
    openFile currentFile
    readFile currentFile for 5000
    cFiletxt = It
    text of field "show text" of page 1 = cFiletxt
    closeFile currentFile
end
```

SYNTAX `openDlgLFN(<dir>,<ext>,<prompt>,<caption>)`

DESCRIPTION Displays a standard Windows Open dialog box.



RETURNS If no error occurs, `openDlgLFN()` returns a string that contains the long file name and path of the file the user specified.

Otherwise, the function returns `null` and sets `sysError` to one of these values:

`-1` User canceled the dialog box.
`-3` Memory allocation error.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING openDlgLFN(STRING,STRING,STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<dir>`	The directory to be displayed. Use `"."` to specify the current directory, or `".."` to specify its parent directory, or include the complete path for the desired directory.
`<ext>`	The extension for the file names to be displayed. For example, `"*.tbk"` displays ToolBook files.
`<prompt>`	The text for the static control at the top of the dialog box, which usually provides instructions to the user.
`<caption>`	The caption for the title bar of the dialog box.

EXAMPLES

```
to handle openText
    get openDlgLFN(".","*.txt","Choose a text file...","Open")
    if sysError is - 1
        break to system
    end if
    currentFile = upperCase(It)
    openFile currentFile
    readFile currentFile for 5000
    cFiletxt = It
    text of field "show text" of page 1 = cFiletxt
    closeFile currentFile
end
```

openFile

File Manipulation Command

SYNTAX

```
openFile <file name>
```

DESCRIPTION

Opens a file.

NOTES

You can open a file to read text from it using the `readFile` command or to append information to the file using the `writeFile` command.

If the specified file doesn't exist, ToolBook sets `sysError` to `no such file`.

A maximum of `10` files can be opened at any one time. Close files as needed to keep this count below `10`.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	A file name, including the path if necessary.

EXAMPLES

```
--Opens a file, with error checking
clear sysError
sysSuspend = false
openFile editFileName
sysSuspend = true
if sysError is not null
    request "File error opening" && editFileName && sysError
    break
end
```

openFileDlg()

TBDLG.DLL File Function (Dialog)

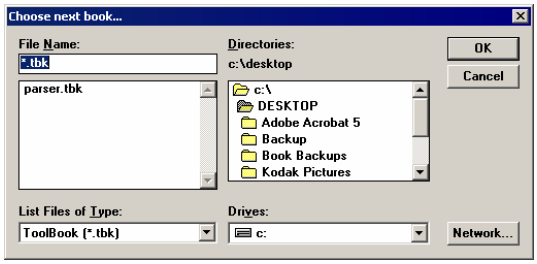
SYNTAX

```
openFileDlg(<caption text>,<default file>,<default path>,<filters>,
  <index>)
```

DESCRIPTION

Displays the standard Windows Open File dialog box.

This function will accept a long file name as input, but will return short file names only. Use `openFileDlgLFN()` if you need to return long file names.



RETURNS

If no error occurs, the function returns a string that contains a file name.

Otherwise, the function returns `null` and sets `sysError` to one of these values:

`-1` User canceled the dialog box.
`-2` Memory allocation error.

NOTES

You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING openFileDlg(STRING,STRING,STRING,STRING,INT)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `openFileDlg32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<caption text>`	The text displayed in the title bar of the dialog box.
`<default file>`	The default file name. If `null`, all files matching the specified filter are shown.
`<default path>`	The default path. If `null`, the current directory is used.
`<filters>`	A list of file name filters. Each filter is a two-item list containing a description and a wildcard, for example "`Icon (*.ico), *.ico`". This parameter can contain multiple pairs of filters.
`<index>`	An index into the `<filters>` list specifying which to use as the default. The first filter is `1`. To get the filter value, use the `getOpenFileDlgFilterIndex()` function.

EXAMPLES

```
to handle buttonClick
    filterList = "Icon (*.ico),*.ico,ToolBook (*.tbk),*.tbk"
    get openFileDlg("Choose next book...",null,null,filterList,2)
    if IT is null
        request "User Canceled."
    else
        request "File selected:" && item 1 of IT
    end
end
```

openFileDlg32()

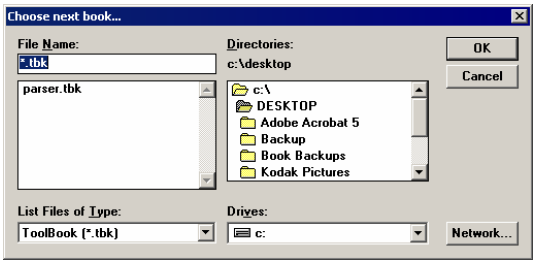
TBFILE32.DLL File Function (Attribute)

SYNTAX

```
openFileDlg32(<caption text>,<default file>,<default path>,<filters>,
  <index>)
```

DESCRIPTION

Displays the standard Windows Open File dialog box.



RETURNS

If no error occurs, the function returns a string that contains a file name.

Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES

As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    STRING openFileDlg32(STRING,STRING,STRING,STRING,INT)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<caption text>`	The text displayed in the title bar of the dialog box.
`<default file>`	The default file name. If `null`, all files matching the specified filter are shown.
`<default path>`	The default path. If `null`, the current directory is used.
`<filters>`	A list of file name filters. Each filter is a two-item list containing a description and a wildcard, for example "`Icon (*.ico), *.ico`". This parameter can contain multiple pairs of filters.
`<index>`	An index into the `<filters>` list specifying which to use as the default. The first filter is `1`. To get the filter value, use the `getOpenFileDlgFilterIndex32()` function.

EXAMPLES

```
to handle buttonClick
    filterList = "Icon (*.ico),*.ico,ToolBook (*.tbk),*.tbk"
    get openFileDlg32("Choose next book...",null,null,filterList,2)
    if IT is null
        request "User Canceled."
    else
        request "File selected:" && item 1 of IT
    end
end
```

openFileDlgLFN()

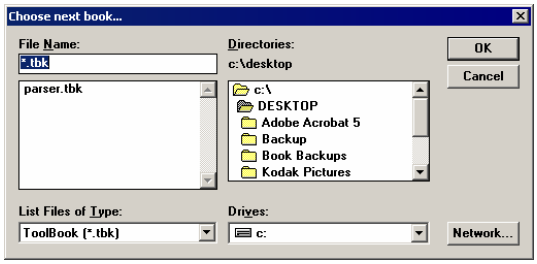
TBDLG.DLL File Function (Dialog)

SYNTAX

```
openFileDlgLFN(<caption text>,<default file>,<default path>,<filters>,
  <index>)
```

DESCRIPTION

Displays the standard Windows Open File dialog box and returns a long file name.



RETURNS

If no error occurs, the function returns a string that contains a file name.

Otherwise, the function returns `null` and sets `sysError` to one of these values:

`-1` User canceled the dialog box.
`-2` Memory allocation error.

NOTES

You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING openFileDlgLFN(STRING,STRING,STRING,STRING,INT)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `openFileDlg32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
<caption text>	The text displayed in the title bar of the dialog box.
<default file>	The default file name. If null, all files matching the specified filter are shown.
<default path>	The default path. If null, the current directory is used.
<filters>	A list of file name filters. Each filter is a two-item list containing a description and a wildcard, for example "Icon (*.ico), *.ico". This parameter can contain multiple pairs of filters.
<index>	An index into the <filters> list specifying which to use as the default. The first filter is 1. To get the filter value, use the getOpenFileDlgFilterIndex() function.

EXAMPLES

```
to handle buttonClick
    filterList = "Icon (*.ico),*.ico,ToolBook (*.tbk),*.tbk"
    get openFileDlgLFN("Choose next book...",null,null,filterList,2)
    if IT is null
        request "User Canceled."
    else
        request "File selected:" && item 1 of IT
    end
end
```

openWindow

Notification Message

DESCRIPTION Sent to a viewer when it is opened.

NOTE As with most all system generated messages, it is important to forward the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle openWindow
    my parentHandle = clientHandle of mainWindow
    forward
end
```

```
to handle openWindow
    my caption = "Date:" & sysDate
    forward
end
```

or

Logic Operators

SYNTAX

```
<expression> or <expression>
```

DESCRIPTION Returns true if the left expression or right expression [or both] evaluates to true. Otherwise returns false.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<expression>	Any expression.

EXAMPLES

```
if score > 85 or teachersPet = true
    request "you passed!"
end
```

outerBevelWidth

Property

DESCRIPTION A stage property that specifies the width of the outer 3D bevel of its frame.

NOTES You can get or set this property.

VALUES A number in `page units`.

The default is `15`.

EXAMPLES
```
outerBevelWidth of stage "player" = 45
```

outline

Property

DESCRIPTION A stage property that specifies whether a thin, black outline is drawn around the stage's frame.

NOTES You can get or set this property.

VALUES `True` or `false`.

The default is `false`.

EXAMPLES
```
outline of stage "player" = false
```

overlayAlias

UNDOCUMENTED Property

DESCRIPTION A property of a stage that specifies an alias for the playback area of the stage, needed when using an overlay device and MCI commands.

NOTES When you use a video overlay device in a stage object, it may be necessary to use an MCI command to set some parameters if the default values do not match your configuration.

The sample script below uses the `overlayAlias` to select video source `1` instead of the default of `0` for a Super Video Windows overlay board:

WARNING THIS IS AN UNDOCUMENTED FEATURE. What that means is that you can use it but you will most likely be unable to get assistance from Click2learn on how to use it, or troubleshoot it.

VALUES An alias reference or `null`.

EXAMPLES
```
overlayOpen of stage "player" = true
x = overlayAlias of stage "Player"
get callMCI("set" && x && "source source1")
```

overlayOpen

Property

DESCRIPTION A stage property that specifies whether a video overlay device is open in the stage's display area.

NOTES You can get or set this property.

To add live video to your application, set a stage's `overlayOpen` property to true. This setting turns the `video overlay` device on and makes the device display the available video in the stage.

VALUES `True` or `false`.

If `true`, a clip that uses `video overlay` is playing in the stage.

EXAMPLES
```
overlayOpen of stage "player" = true
x = overlayAlias of stage "Player"
get callMCI("set" && x && "source source1")
```

DESCRIPTION The object type name for a `page`, which includes the foreground with a reference by name or `ID` to its background.

NOTES The parent of a page is its background.

To create a page, use the `send newPage` statement.

TO REFER TO A PAGE	USE THIS SYNTAX
Currently displayed	`this page`
Of a background	`page "summary" of background "history"`
In the current book	`page "Magnolia"` `page ID 43`
In a viewer	`currentPage of viewer "palette"` `currentPage of viewer ID 2`
In another book	`page "Alder" of book "landscape.tbk"`

In a position relative to the current page	`next page`
	`previous page [of this book]`
	`first page of this background`
	`last page of book "landscape.tbk"`

By its ordinal position	`first page of background "Trees"`
	`fifth page of this book`
	`tenth page of book "landscape.tbk"`

By page number	`page 40 of this background`
	`page 37 of this book`
	`page 5 of book "landscape.tbk"`

All individual `menu event` messages, including those for custom menu items, and all `DDE` messages are sent to the page with the exception of the `menuItemSelected` and `enterMenu` messages are sent to the viewer that owns the menu bar. Messages sent to recordfields are not forwarded to pages because recordfields are on backgrounds.

The following built-in and enter event and leave event messages are sent to the current page: `activateInstance`, `enterBackground`, `enterBook`, `enterPage`, `enterSystem`, `leaveBackground`, `leaveBook`, `leavePage`, and `leaveSystem`. The `enterApplication` and `leaveApplication` messages are sent only to the ToolBook Main window just before the `enterBook` or just after the `leaveBook` message is sent.

You can automatically forward messages that are handled by the page to objects anywhere in the hierarchy by using `notifyBefore` or `notifyAfter` handlers in their scripts. For example, if you want an object on a page to receive the `enterPage` message before the page receives it, you could place a `notifyBefore enterPage` handler in that object's script.

You can use the term `pages` with `flip`, `print`, `printerStyle`, and `sort` to indicate pages in a book, as in `flip 5 pages`.

The size of a page is defined by its background's `size` property.

Place an `enterPage` handler in a page script if you want that handler to execute each time ToolBook displays the page. Place other handlers in the page script if you want a message to be handled the same way for several objects on the page.

To delete a page, `send selectPage`, then `send clear`.

You can show a page from another book without opening the other book by displaying the page in a viewer. To display a page from another book without opening that book's instance, set the `currentPage` property of a viewer to a valid page reference in another book.

pageCount

Property

DESCRIPTION A book property that specifies the number of pages in a book.
A background property that specifies the number of pages that use that background.

NOTES You cannot set this property.

VALUES A positive whole number representing the number of pages.

EXAMPLES

```
-- ensure every page is named
step k from 1 to pageCount of this book
    if name of page k = null
        name of page k = ("Page#" & k)
    end
end

-- clear the recordfield text from each page
bgnd = background "class"
step k from 1 to pageCount of bgnd
    text of recordfield "comments" of page k of bgnd = null
end
```

pageFromClient()

TBWIN.DLL Screen Display Function

SYNTAX `pageFromClient(<pageScroll>,<magnification>,<point|rectangle>)`

DESCRIPTION Converts a set of coordinates in `pixels` relative to the upper-left corner of the `client area` of a ToolBook window into coordinates in ToolBook `page units`. The client area is defined as the working area below the menu bar, not including the scroll bars.

RETURNS If no error occurs, `pageFromClient()` returns the coordinates of either a `point` or a `rectangle`, depending on the last parameter.

Otherwise, the function returns `null` and `sysError` is set to one of these values:

`-20` Memory allocation error.
`-99` First or last parameter was invalid.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    STRING pageFromClient(STRING,INT,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<pageScroll>`	The value of `pageScroll` for a viewer such as the `mainWindow`, this window, or a valid viewer reference.
`<magnification>`	The value of `magnification` for a viewer such as the mainWindow, this window, or a valid viewer reference.
`<point|rectangle>`	For a `point`, a list containing two numbers (x and y coordinates). For a `rectangle`, a list containing four numbers (coordinates of upper-left and lower-right corners of the rectangle).

EXAMPLES

```
-- Draw a rectangle on the page, outlining the border where the child
-- viewer is currently sitting.
in mainWindow
    draw rectangle from 0,0 to 1,1
    bounds of Selection = pageFromClient(pageScroll of mainWindow, \
      magnification of mainWindow,bounds of viewer "help")
end
```

pageFromScreen()

TBWIN.DLL Screen Display Function

SYNTAX `pageFromScreen(<windowHandle>,<pageScroll>,<magnification>, <point|rectangle>)`

DESCRIPTION Converts a set of coordinates in `pixels` relative to the upper-left corner of the `display screen` into coordinates in ToolBook `page units`.

RETURNS If no error occurs, `pageFromScreen()` returns the coordinates of either a `point` or a `rectangle`, depending on the last parameter.

Otherwise, the function returns `null` and `sysError` is set to a negative value:

`-20` Memory allocation error.

`-30` `<windowHandle>` was invalid.

`-99` Second or last parameter was invalid.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    STRING pageFromScreen(WORD,STRING,INT,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<windowHandle>`	The value of `windowHandle` of a viewer such as the `mainWindow`, this window, or a valid viewer reference.
`<pageScroll>`	The value of `pageScroll` for a viewer such as the `mainWindow`, this window, or a valid viewer reference.
`<magnification>`	The value of `magnification` for a viewer such as the mainWindow, this window, or a valid viewer reference.
`<point|rectangle>`	For a `point`, a list containing two numbers (x and y coordinates). For a `rectangle`, a list containing four numbers (coordinates of upper-left and lower-right corners of the rectangle).

EXAMPLES

pageNumber

Property

DESCRIPTION A page property that specifies the page number of a page in a book.

NOTES You can get or set this property.

Changing the value of this property changes the order of pages in the book.

VALUES A positive whole number.

EXAMPLES

```
-- move page "r11" to the 2nd page position in the book
pageNumber of page "r11" = 2
```

pages

Property

DESCRIPTION A book property that holds a list of all the pages in a book.

A background property that holds a list of all the pages used by that background.

NOTES This is a read only property and cannot be set.

VALUES A comma separated list of page references.

EXAMPLES

```
Both of these routines do the same thing with different approaches

-- Navigate to the 17th page used by background "x3"
go to (item 17 of pages of background "x3")

-- Navigate to the 17th page used by background "x3"
go to page 17 of background "x3"
```

pageScroll

Property

DESCRIPTION A property of a viewer that specifies the offset of the page from the upper-left corner of the client window in which it is displayed.

NOTES You can get or set this property.

Each viewer can be scrolled differently, so the setting for the `pageScroll` property of each viewer can be a different value.

The value of `sysPageScroll` and the value of `pageScroll` for the mainWindow are one in the same. Changing the value of one automatically updates the other setting.

VALUES A list of two positive whole number in `page units` representing the horizontal and vertical coordinates of the point on the page that is located in the upper-left corner of the client window.

The coordinates can only be set to make the page fill the client window.

The default value is `0,0`.

EXAMPLES

```
pageScroll of viewer "pan-n-scan" = 40,40
```

pageScrolled

Notification Message

SYNTAX

```
pageScrolled <old pageScroll>,<new pageScroll>
```

DESCRIPTION Sent to a viewer when the `pageScroll` of the viewer changes.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
`<old pageScroll>`	A list of two numbers in `page units` that represent the value of the pageScroll property before the scroll occurs.
`<new pageScroll>`	A list of two numbers in `page units` that represent the value of pageScroll after the scroll occurs.

EXAMPLES

```
-- Adjusts position of non-tiled child window to new scroll position
to handle pageScrolled
    -- pageUnitsToClient automatically accounts for scroll position
    position of viewer "nested view" = \
      pageUnitsToClient(position of rectangle "place holder")
    forward
end
```

pageUnitsToClient()

Screen Display Function

SYNTAX

`pageUnitsToClient(<coordinates>)`

DESCRIPTION

Converts `page units` to `pixels` relative to the origin of the client window.

Use this function to align a child window of a viewer's client window with an object on the page. This function accounts for the current values of the viewer's `pageScroll` and `magnification` properties.

NOTES

The conversion of `page units` to client-relative `pixels` is dependent upon the currently installed video driver.

PARAMETERS

PARAMETER	DESCRIPTION
`<coordinates>`	A list of numbers in `page units` that specify a position or positions on the page. You can include as many numbers as necessary, but they must be in pairs. You can refer to the `bounds`, `position`, `size`, or `vertices property` as the value for `<coordinates>`.

pageUnitsToFrame()

Screen Display Function

SYNTAX

`pageUnitsToFrame(<coordinates>,<viewer reference>)`

DESCRIPTION

Converts `page units` for a specified viewer into `pixels` that are relative to the origin of the `client area` of the viewer's frame window.

This function accounts for the current values of the viewer's `pageScroll` and `magnification` properties.

NOTES

Use this function to align a window that is a child of a viewer's frame window with an object on the page displayed in the viewer. (A tiled viewer is an example of a viewer this is the child of a viewer's frame window.) You can also use this function to determine whether the client window is offset from the origin of the frame window's client area.

When a viewer's `centerClient` property is `true`, the client window will be centered in the frame window's client area if the frame window is larger than the client window.

The conversion of `page units` to window-relative `pixels` is dependent upon the currently installed video driver.

PARAMETERS

PARAMETER	DESCRIPTION
`<coordinates>`	A list of numbers in `page units` that specify a position or positions on the page. You can include as many numbers as necessary, but they must be in pairs. You can refer to the `bounds`, `position`, `size`, or `vertices property` as the value for `<coordinates>`.
`<viewer reference>`	A valid viewer reference. If the specified viewer is not open, ToolBook displays an error message.

pageUnitsToScreen()

Screen Display Function

SYNTAX `pageUnitsToScreen(<coordinates>,<viewer reference>)`

DESCRIPTION Converts `page units` for a specified viewer into `pixels` that are relative to the origin of the screen.

Use this function to align a popup window with an object on a page displayed in the viewer. This function accounts for the values of the viewer's pageScroll and magnification properties.

NOTES The conversion of `page units` to screen-relative `pixels` is dependent upon the currently installed video driver.

PARAMETERS

PARAMETER	DESCRIPTION
`<coordinates>`	A list of numbers in `page units` that specify a position or positions on the page. You can include as many numbers as necessary, but they must be in pairs. You can refer to the `bounds`, `position`, `size`, or `vertices property` as the value for `<coordinates>`.
`<viewer reference>`	A valid viewer reference. If the specified viewer is not open, ToolBook displays an error message.

paintObject

Object Type

DESCRIPTION The object type name for a bitmap graphics that are pasted or imported into ToolBook from an original graphic format that was either bitmap (.BMP) or device-independent bitmap (.DIB).

NOTES A paint object's parent can be a page, background, or group if the graphic is in a group.

To import a paint object using OpenScript, use the `importGraphic` command.

When you import a bitmap into ToolBook, make sure it's the size you want because paint objects cannot be resized on the page. If you attempt to resize a paintObject you will effectively end up cropping the image.

If you cut or copy a bitmap from an `OLE server` application and paste the bitmap on a page, the bitmap is pasted as an `OLE` object. To paste a bitmap from an `OLE server` application as a paint object, copy it from the source application, choose `Paste Special` from the Edit menu, then select the `Bitmap` option.

You can change the stroke color of a paint object's border, but not its fill color. To remove its border, set its `lineStyle` to `none`.

You can convert a `picture` to a `paint object` by clicking the `convert` button in the object's Properties dialog box.

palette

Resource Type

DESCRIPTION The resource type name for a `color palette` resource.

Color palette resources are referenced in OpenScript by `name` or by `ID` using the `palette` keyword. The `ID` is determined by the ToolBook system, but you can assign any `name` to the resource.

You can set a palette resource for the `palette` property of a book.

EXAMPLES

```
palette of this book = palette "custom"
palette of this book = "c:\temp\custom.pal"
```

palette

Property

DESCRIPTION A book property that specifies the `color palette` associated with the book.

NOTES You can get or set this property.

When a color palette is assigned to a book, ToolBook uses the palette to display all the book's paint objects and pictures, which helps eliminate palette shift and makes page navigation faster.

The exception to using one color palette for the book is that some images display better when their native color palette is used.

The use of color palettes is pretty much a holdover from the days when computers commonly were configured to only display `256` [`8-bit`] colors. If configured to display `16-bit`, `24-bit`, or higher color, then the palette is not utilized for painting the images on the screen.

VALUES A valid file name or reference to a palette resource, or `null`.

If `null`, ToolBook uses the Windows default system colors.

Getting the `palette` property returns the resource reference.

EXAMPLES

```
palette of this book = "c:\media\project1.pal"
palette of this book = palette "project#1" of book "c:\template.tbk"
```

parent

Property

DESCRIPTION The `parent` property specifies the object's parent, which is the object immediately above it in the `object hierarchy`.

NOTES You can get but you cannot set this property.

The `parent` object of a `hotword` is a `field` or `recordfield`.

For an object in a `group`, the `parent` is the group.

For an ungrouped object, the parent is the `page` or `background` on which the object is located.

For a page, the `parent` is a `background`.

For a `background`, the `parent` is a `book`.

A `viewer's` parent object is always the `book`.

VALUES The `parent` property holds the `uniqueName` of the parent object.

EXAMPLES

```
to handle buttonClick
    move parent of self by 400,600
end
```

parentHandle

Property

DESCRIPTION A property of a viewer that specifies the `16-bit` window handle of a viewer's parent window.

NOTES You can get or set this property for open viewers.

When a viewer is first opened, its parent window is the target window.

You can change a viewer's parent window by setting its `parentHandle` or `parentWindow` property.

You can determine whether a viewer has a parent window by getting its `parentHandle`.

You can change the viewer's parent window by setting this property to the window handle of any existing window on the desktop.

VALUES A valid `16-bit` window handle or `0`.

The default is the value of `targetWindow`.

Setting `parentHandle` to `0` specifies the Windows desktop as the parent window.

You can set a viewer's `parentHandle` to the handle of a window in another application or in another ToolBook instance.

EXAMPLES

```
-- ensure both viewers have the same parent
parentHandle of viewer "help" = parentHandle of viewer "forms"
```

parentHandle32

Property

DESCRIPTION A property of a viewer that specifies the `32-bit` window handle of a viewer's parent window.

NOTES You can get or set this property for open viewers.

When a viewer is first opened, its parent window is the target window.

You can change a viewer's parent window by setting its `parentHandle32` or `parentWindow` property.

You can determine whether a viewer has a parent window by getting its `parentHandle32`.

You can change the viewer's parent window by setting this property to the window handle of any existing window on the desktop.

VALUES A valid `32-bit` window handle or `0`.

The default is the value of `targetWindow`.

Setting `parentHandle32` to `0` specifies the Windows desktop as the parent window.

You can set a viewer's `parentHandle32` to the `32-bit` handle of a window in another application or in another ToolBook instance.

EXAMPLES

```
-- ensure both viewers have the same parent
parentHandle32 of viewer "help" = parentHandle32 of viewer "forms"
```

parentWindow

Property

DESCRIPTION A property of a viewer that specifies the viewer's parent window.

NOTES You can get or set this property.

When a viewer is first opened, its parent window is set to the target window.

You can change a viewer's parent window by setting its `parentWindow` or `parentHandle` property.

VALUES A valid reference to a viewer, or `null`.

The default is the value of `targetWindow`.

If `null`, the parent window is either the Windows desktop or a non-ToolBook window.

When `parentWindow` is `null`, use the `parentHandle` property to determine the parent window.

EXAMPLES

```
-- ensure both viewers have the same parent
parentWindow of viewer "help" = parentWindow of viewer "forms"
```

paste

Clipboard Message

DESCRIPTION Sent to the page when `Paste` is chosen from the `Edit` menu.

You can also `send` this message using the `send paste` statement. ToolBook's default response is to paste the contents of the `Clipboard` into the current book or, if the Clipboard is empty, to do nothing.

To paste an object using OpenScript, set `focus` to `null` and `send` the `paste` message.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle paste
    focus = null
    forward
end
```

pasteSpecial

Clipboard Message

SYNTAX `pasteSpecial [<clipboard format>]`

DESCRIPTION Sent to the page when you choose `Paste Special` from the `Edit` menu.

You can also `send` this message using the `send pasteSpecial` statement. If no `<clipboard format>` parameter is specified, ToolBook displays the Paste Special dialog box. The Paste Sp ecial dialog box contains a list of supported data formats you can select from to specify the format of the data pasted from the Clipboard.

When the `<clipboard format>` parameter is specified, ToolBook pastes the contents of the Clipboard in the specified format without displaying the Paste Special dialog box. If the specified format is invalid, ToolBook sets `sysError` to "`Invalid Clipboard Format`".

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
`<clipboard format>`	A string that specifies the format of the data to be pasted. This parameter is optional.

EXAMPLES

```
send pasteSpecial "picture"
send pasteSpecial "text"
send pasteSpecial "rich text format"
```

pattern

Property

DESCRIPTION A draw object or background property that specifies the current fill.

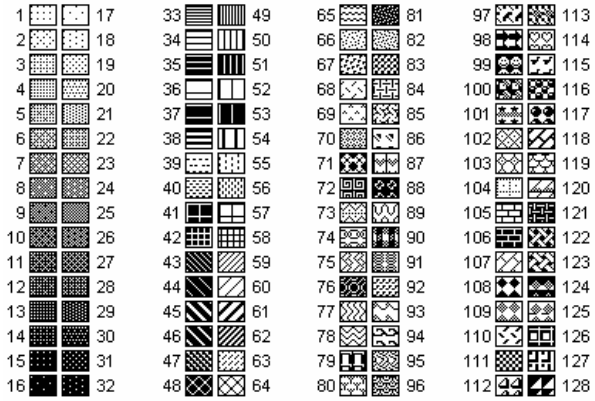
NOTES You can get or set this property.

For solid color objects, set this property to `solidStroke` or `solidFill`, rather than `none`. If set to `none`, the objects appear white and transparent.

VALUES This property can contain the following values:

`None`, `solidFill`, `solidStroke`, or an whole number in the range from `1` to `128`.

The default pattern is the value of `sysPattern` when the object was created.



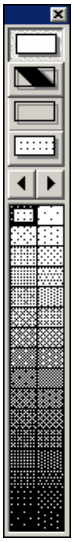
EXAMPLES

```
pattern of this background "apple" = 125

pattern of rectangle "cardoor" = solidFill

-- trick to force the page to redraw by re-applying
-- the background pattern.
pattern of this background = pattern of this background
```

patternPalette

System Object

DESCRIPTION The object type name for the `pattern palette`, which is used to specify a pattern to fill enclosed draw objects or the background.



NOTES To open the pattern palette, use the `show patternPalette` statement in a script. You can also show or hide the pattern palette by clicking the Pattern palette button on the tool bar.

You cannot show the pattern palette in Runtime ToolBook.

Use the `hide`, `show`, and `move` commands to control the visibility or position of the pattern palette.

PROPERTY	VALUES
`bounds`	List of four whole numbers in `pixels`.
`position`	List of two whole numbers in `pixels`.
`vertices`	List of four whole numbers in `pixels`.
`visible`	`True` or `false`.

pause

Timer Function Script Control

SYNTAX

```
pause <time> [ticks]
pause <time> seconds
```

DESCRIPTION Causes ToolBook to wait before executing the rest of the script.

If `seconds` are not specified, `ticks` are used. A `tick` is approximately `1/100` of a second.

NOTES The `pause` command freezes any operation on the computer and prevents any other events or messages from occurring for the duration of the pause.

PARAMETERS

PARAMETER	DESCRIPTION
<time>	A number.

EXAMPLES

```
to handle buttonClick
    show field "help"
    pause 3 seconds
    hide field "help"
end
```

percentFreeSpace

Property

DESCRIPTION

A page or background property that specifies the percentage of free memory available on a page or background.

NOTES

You cannot set this property.

When objects are deleted from a page or background, the resources they used are not automatically freed. To free the space, choose `Save As` from the `File` menu and save the current book to a new file name. The book is compacted to its minimum size and any available space is freed on pages and backgrounds.

Each page and background can hold a maximum of `64K` of data, so the return value for `percentFreeSpace` indicates the largest block of free memory on the page or background as a percentage of `64K`.

VALUES

A number representing the percent free space ranging from `0` to `100`.

EXAMPLES

```
freeSp = percentFreeSpace of background "apple"
```

```
-- Draw 10 rectangles
step k from 1 to 10
    if percentFreeSpace of this page > 5
        -- enough space to create a rectangle
        draw rectangle from 0,0 to 500,500
    else
        -- running out of space, warn user
        request "Could not complete operation."
    end
end
```

pi

Special Constants

DESCRIPTION

Represents the mathematical value of pi. This is equivalent to `3.141592653589793`.

ACTIONS EDITOR

The Actions Editor also supports this feature.

EXAMPLES

```
radius = 53
areaOfCircle = pi * radius ^ 2
```

picture

Object Type

DESCRIPTION The object type name of a graphic object that is pasted into ToolBook in Windows metafile format (.WMF), or a picture file (.DRW, .EPS, .CGM, or .TIF) that is imported into ToolBook.

NOTES A picture's parent can be a page, background, or group if the picture is in a group.

To create a picture using OpenScript, use the `importGraphic` command.

If you cut or copy a picture from an `OLE server` application and paste the picture on the page, it is pasted as an `OLE` object. To paste a graphic that is created in an `OLE server` application as a picture, copy it from the source application, choose `Paste Special` from the Edit menu, then select the `Picture` option.

You can convert a `picture` to a `paint object` by clicking the `Convert` button in the object's Properties dialog box.

You can change the stroke color of a picture's border, but not its fill color. To remove its border, set its `lineStyle` to `none`.

pie

Object Type

SYNTAX `draw pie from <location> to <location> to <location>`

DESCRIPTION The object type name for a `pie`.

NOTES A pie's parent can be a page, background, or group if the pie is in a group.

For a rounded rectangle, the `bounds` and `vertices` properties are not equal.

playSound()

Sound Commands

SYNTAX `playSound(<sound>[,<wait>])`

DESCRIPTION Plays a wave audio (`.WAV`) file.

If the function returns `true`, the file plays successfully and any other wave file currently playing is preempted.

The function returns `false` if an error occurs.

This function does not notify ToolBook of its status.

NOTES The `playSound()` function works best with files less than `100K` in size. If you need to play larger files, it is recommended that you use a `Clip` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<sound>`	A string that represents a valid `.WAV` file. If `null`, the function stops the sound file that is currently playing if allowed by the driver.
`<wait>`	`True` or `false`; the default is `false`. If `false`, the function returns immediately and script execution continues while the sound plays. If `true`, the function waits until the wave file has finished playing before it returns.

EXAMPLES

```
to handle buttonClick
    get playSound("c:\project\cardoor.wav")
end
```

pmt()

Financial Values

SYNTAX `pmt(<rate>,<nper>,<present value>,<future value>[,<type>])`

DESCRIPTION A financial function that returns the periodic payment for an annuity. This function is based on payments made regularly at a constant interest rate. The `pmt()` function calculates annuity due and accounts for the future value of the principal.

PARAMETERS

PARAMETER	DESCRIPTION
`<rate>`	A number representing the interest rate per period.
`<nper>`	A positive whole number representing the total number of payment periods.
`<present value>`	A number representing the present value, or the lump-sum amount that a series of future payments is worth at the present time.
`<future value>`	A number representing the future value, or the cash balance after the last payment is made. If `<future value>` is omitted, it is assumed to be `0`.
`<type>`	Optional Parameter. A value of `true` or `false` that indicates when payments are due. `True` indicates that payments are due at the beginning of the period (annuity due), `false` at the end (ordinary annuity). If `<type>` is omitted, it is assumed to be `false`.

EXAMPLES

```
-- Shows monthly payment for 30 year $150,000 loan at 7.5% interest
x = pmt(.075/12,12*30,150000,0,false)
```

polygon

Object Type

SYNTAX `draw polygon from <location> to <location>`

DESCRIPTION The object type name for a `polygon`.

NOTES A polygon's parent can be a page, background, or group.

The `bounds` and `vertices` properties of a polygon don't have the same values.

To reshape a polygon, select the polygon and send the `reshape` message, then drag the reshape handles to a new position.

To add a vertex to a polygon, select the polygon and send the `reshape` message. Press the `Shift` key as you click a vertex handle, and a new vertex handle appears adjacent to the existing handle. Then drag the reshape handle to a new point. To remove a vertex from a polygon, select the polygon and send the `reshape` message. Press `CTRL-Shift` as you click the vertex handle.

Though the polygon may appear to be an irregular polygon after you reshape, add, or remove its vertices, ToolBook stills references it as a `polygon`.

polygonPalette

System Object

DESCRIPTION The object type name for the `polygon palette`, which is used to specify the number of sides for polygons created with the polygon tool.

NOTES To open the polygon palette, use the show `polygonPalette` statement in a script.

You can also show or hide the polygon palette by clicking the Polygon palette button on the tool bar.

You cannot show the polygon palette in Runtime ToolBook.

Use the `hide`, `show`, and `move` commands to control the visibility or position of the polygon palette.

PROPERTY	VALUES
`bounds`	List of four whole numbers in `pixels`.
`position`	List of two whole numbers in `pixels`.
`vertices`	List of four whole numbers in `pixels`.
`visible`	`True` or `false`.

pop

Data Manipulation

SYNTAX `pop [<stack>] [into | before | after <container>]`

DESCRIPTION Retrieves the first item of a `stack`, which can be any list of items.

If a `stack` is not specified, the item is popped from `sysHistory`.

If no container is specified, the popped item goes into `IT`.

NOTES The `pop...into` form replaces the contents of the container with the value of the popped item.

The `pop...before` form inserts the value of the popped item before the container contents.

The `pop...after` form appends the value of the popped item after the container contents.

After an item has been popped, it is removed from the `stack`, and the number of items in the `stack` is reduced by one. If you try to pop an item from an empty `stack`, an error occurs and ToolBook displays an `Execution Suspended` message.

When popping items in a stack, use these guidelines:

- ToolBook pops items in reverse of the order that they were pushed.
- If speed is important, use pop rather than refer to "`item of`" a stack.
- To find out if a stack is empty, test the list for `null`.

PARAMETERS

PARAMETER	DESCRIPTION

<stack>	A list of items; the default is the value of sysHistory.
	Note When a local stack variable is passed as a parameter, it is side effected in the set handler. For example:
	`to handle buttonClick`
	`  local stack temp`
	`  temp = "foo"`
	`  testProp() = temp`
	`  request temp`
	`end`
	
	`to set testProp to testArg`
	`  pop testArg`
	`end`

<container>	The name of the destination container; the default is It.

EXAMPLES

```
-- name all unnamed backgrounds.
bgs = backgounds of this book
while bgs <> null
    pop bgs into curBgnd
    if namd of curBgnd = null
        name of curBgnd = "unnamed"
    end
end
```

popupMenu()

Menu Command/Function

SYNTAX

popupMenu(<location>,<resourceRef>,<menu name>)

DESCRIPTION

Displays a popup menu for a menu bar resource at a specified location.

NOTES

When the user chooses a menu item, the popup menu is dismissed and the function returns a list with the text of the selected menu item and its alias. The user can also click anywhere outside the menu to close it. If no menu item is selected, popupMenu() returns null.

You can use the name and alias returned by the function to specify the action that occurs as a result of the user's choice.

PARAMETERS

PARAMETER	DESCRIPTION
<location>	A list of two numbers specifying the location of the upper-left corner of the popup menu on the page or background.
<resourceRef>	A valid reference to a menu bar resource.
<menu name>	Specifies the menu on the menu bar resource to serve as the popup menu. When the value is null, ToolBook uses the first menu of the specified menu bar resource.

EXAMPLES

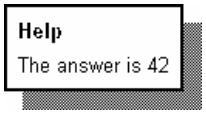
```
-- Displays the Help menu from the menu bar resource "custom" one inch
-- from the upper left corner of page
get popupMenu("1440,1440", menuBar "custom", "Help")

-- Creates a right-click menu at Reader level for the selected object
to handle rightButtonDown pLoc
    get popupMenu(pLoc, menuBar first word of target, null)
    send (item 2 of It)       --Parentheses force evaluation of It
end

-- Uses the alias returned by the function to go to a page specified
-- by the user
to handle rightButtonDown pLoc
    get popupMenu(pLoc, menuBar "GoToPage", null)
    go to page item 2 of It
end
```

popText()

TBWIN.DLL Dialog Box

SYNTAX `popText(<title>,<text string>,<point>)`

DESCRIPTION Displays text in a read-only, popup window with a shadowed border.



RETURNS If successful, the function returns `0`.

Otherwise, the function returns this error:

`-1` Unable to open the window.

NOTES The window is sized appropriately for the text displayed and the position is adjusted if necessary to keep the window on the screen.

The text is displayed in the Arial font face.

The popup window is dismissed when the user presses a key or clicks the mouse.

Unlike `modalPopText()`, the use of `popText()` does not halt script execution while the popup window is displayed.

As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT popText(STRING,STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<title>`	A string that specifies the text to be displayed in bold above the text string in the popup window.
`<text string>`	A string that specifies the text to be displayed in the popup window.
`<point>`	A string containing a list of two numbers in `page units` that represent the upper-left corner of the popup window.

EXAMPLES

```
to handle buttonClick
    get popText("Help","The answer is 42",sysmouseposition)
end
```

popTextGetBounds()

TBWIN.DLL Dialog Box

SYNTAX `popTextGetBounds(<title>,<text string>,<point>)`

DESCRIPTION Returns the bounds of the popup window to be displayed by the `popText()` function, based on the specified values for text and point.

RETURNS A list of four numbers in `page units` that represent the bounds of the popup window. (The bounds can be outside the bounds of the ToolBook window.)

Otherwise, the function returns this error:

`-1` Unable to perform the operation.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT popTextGetBounds(STRING,STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<title>	A string that specifies the text to be displayed in bold above the text string in the popup window.
<text string>	A string that specifies the text to be displayed in the popup window.
<point>	A string containing a list of two numbers in page units that represent the upper-left corner of the popup window.

EXAMPLES

```
to get msgInfo cap, txt, pos
    return popTextGetBounds(cap,txt,pos)
end
```

position

Property

DESCRIPTION A property of fields, recordfields, groups, buttons, comboboxes, draw objects, ole objects, paint objects, picture objects, stages, viewers and system objects which specifies the screen position of the object.

NOTES You can get or set this property.

For viewers, you can only get the value of position for a viewer that is open.

VALUES A list of two numbers that define the location of the upper-left corner of the object.

For viewers the position is in pixels and relative to the upper-left corner of the screen.

For fields, recordfields, groups, buttons, comboboxes, draw objects, ole objects, paint objects, picture objects, and stages, the position is in page units and relative to the upper-left corner of the client area of the ToolBook window.

EXAMPLES

```
position of button "apple" = 0,0
```

```
position of button "apple" = position of rectangle "orange"
```

```
position of viewer "passwordDialog" = 50,50
```

postEffect

Property

DESCRIPTION A stage property that specifies the transition effect that occurs just after its media finishes playing.

NOTES You can get or set this property.

VALUES A string of options that describe the transition effect. Each option is separated by at least one space.

The string can include the type of effect, the direction of the effect's movement, the destination of the effect's movement, and the speed of the transition.

The effect string uses the following syntax:

<effect> <direction> <destination> <speed>

Use the following table to determine the possible settings for effect, direction and destination.

EFFECT	DIRECTION	DESTINATION
blinds		
dissolve		
drip		
fade		
iris		
push		left, right, top, bottom
puzzle		
rain		left, right
slide	in, out	left, right, top, bottom
spiral	in, out	
split	in, out	horizontal, vertical
tear		left, right, top, bottom, horizontal, vertical
turnPage		left, right
wipe		left, right, top, bottom
zoom	in, out	left, right, bottom, top, center, lowerLeft, lowerRight, upperLeft, upperRight

The value for `speed` can be: `slow`, `normal`, `fast`, or `speed <milliseconds>`.

EXAMPLES

```
postEffect of stage "player" = "split out horizontal fast"
postEffect of stage "player" = "split out horizontal speed 2000"
postEffect of stage "player" = "blinds"
```

ppmt()

Financial Values

SYNTAX

```
ppmt(<rate>,<period>,<nper>,<present value>[,<future value>][,<type>])
```

DESCRIPTION

A financial function that returns the payment on the principal for an investment (or borrowed sum) for a given period. This function is based on periodic, fixed payments and a constant interest rate.

PARAMETERS

PARAMETER	DESCRIPTION
<rate>	A positive whole number representing the interest rate per period.
<period>	A positive whole number representing the period for which the interest is to be calculated. The <period> should be in the range from 1 to <nper>.
<nper>	A positive whole number representing the total number of payment periods.
<present value>	A number representing the principal, or the lump-sum amount borrowed.
<future value>	A number representing the cash balance after the last payment is made. If <future value> is omitted it is assumed to be 0. This parameter is optional.
<type>	Optional Parameter. A value of `true` or `false` that indicates when payments are due. `True` indicates that payments are due at the beginning of the period (annuity due), `false` at the end (ordinary annuity). If <type> is omitted it is assumed to be `false`.

EXAMPLES

```
-- Calculates principal paid for month 20 of a 30-year
-- (12 periods per year) loan of $159,000 at 7.375% interest
x = ppmt(.07375/12,20,30*12,159000,0,false)
```

preEffect

Property

DESCRIPTION A stage property that specifies the transition effect that occurs just before its media starts playing.

NOTES You can get or set this property.

VALUES A string of options that describe the transition effect. Each option is separated by at least one space.

The string can include the type of `effect`, the `direction` of the effect's movement, the `destination` of the effect's movement, and the `speed` of the transition.

The effect string uses the following syntax:

```
<effect> <direction> <destination> <speed>
```

Use the following table to determine the possible settings for `effect`, `direction` and `destination`.

EFFECT	DIRECTION	DESTINATION
blinds		
dissolve		
drip		
fade		
iris		
push		left, right, top, bottom
puzzle		
rain		left, right
slide	in, out	left, right, top, bottom
spiral	in, out	
split	in, out	horizontal, vertical
tear		left, right, top, bottom, horizontal, vertical
turnPage		left, right
wipe		left, right, top, bottom
zoom	in, out	left, right, bottom, top, center, lowerLeft, lowerRight, upperLeft, upperRight

The value for `speed` can be: `slow`, `normal`, `fast`, or `speed <milliseconds>`.

EXAMPLES

```
preEffect of stage "player" = "split out horizontal fast"
preEffect of stage "player" = "split out horizontal speed 2000"
preEffect of stage "player" = "blinds"
```

previous

Navigation Message

DESCRIPTION Sent to the page when `Previous Page` is chosen from the `Go` menu.

You can also send this message using the `send previous` statement. ToolBook's default response is to display the page before the current page in a book.

If `previous` is sent when the user is on the first page of a book, ToolBook displays the last page of the book.

NOTE: Please see `skipNavigation` to see how this page property can affect your attempts to navigate.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle buttonClick
    send previous
end

to handle previous
    request "Sorry, you are not permitted to leave this page yet."
end
```

print

Printing

SYNTAX

```
print [<number> [pages]]
```

DESCRIPTION

Prints pages or reports starting with the current page.

You can print all pages of a book, or selected pages. For reports, you can print information from all pages that share a background, or print from selected pages.

This command must be used within a `start spooler` control structure.

To begin printing from a specific page of a book, specify the starting page using the `go` command within the start spooler control structure. In other words you must navigate to the page you want to start printing from.

If a number of pages is specified, ToolBook uses that number to set a range of pages to be printed. If the `printerConditions` property is set to conditions that certain pages in the range do not satisfy, ToolBook will not print those pages.

If the number of pages specified is larger than the number from the current page to the last page, ToolBook stops printing after the last page of the book. If all the pages are to be printed, printing begins with the current page through the last page, then continues from the first to the current page.

You can set various printer properties to customize printing. To print a report, you must set the `printerStyle` property to `columns` or `groups`.

PARAMETERS

PARAMETER	DESCRIPTION
`<number>`	The word "`all`" or a non-negative integer indicating the number of pages to be printed, beginning with the current page. To ensure that all pages from the current page to the end of the book are printed, use a large number for `<number>` even if that number exceeds the page count of the book.

EXAMPLES

```
to handle censusReport
    printerStyle = columns
    printerMargins = 360,495,1440,1440
    printerGutters = 15,60
    printerFields = "Name,First Name,Sex,Spouse Name,Children"
    printerFieldWidths = "1680,1545,674,1739,1167"
    printerFieldNames = true
    printerClipText = false
    printerConditions = "idNumber of this Background = 0"
    start spooler
        print all
    end
end

-- Print the current page
to handle buttonClick
    start spooler
        print 1
    end
end
```

DESCRIPTION Advances the printer to the top of the next sheet. This is useful for printing a blank sheet of paper.

This command must be used within a `start spooler` control structure.

EXAMPLES

```
start spooler
    print 1 pages
    print eject
end spooler
```

Printing Properties

DESCRIPTION The following table provides descriptions and values for all printer system properties.

NOTES ToolBook printer system properties

PRINTER PROPERTY	DESCRIPTION	VALUE
`printerArrangement`	Specifies the number of pages to be printed per sheet.	A list of two positive integers indicating the number of pages to be printed across and down a sheet; default is 1,1.
`printerBorders`	Specifies that pages are printed with a border. Setting this property is the same as choosing `Print Borders Around Pages` in the `Print Page Options` dialog box.	`True` or `false`; default is `true`.
`printerBottomMargin`	Specifies the width of the bottom margin on the printed sheet.	A number in `page units`; default is `1440` (`1"`).
`printerClipText`	Specifies that text is wrapped or clipped in reports at the right and bottom column or group boundaries, using same wordwrap specifications as in fields. Text can be clipped between any two characters, allowing the maximum amount of text printed. Set this property to `true` to print mailing labels and preprinted forms.	`True` if text is clipped; `false` if text wraps; default is `false`.
`printerConditions`	Specifies the conditions under which fields or pages are printed, using any OpenScript operators plus `text`, `date`, and `name` comparisons. Setting this property is the same as specifying a `Pages Where` condition in the `Print Pages` or `Print Report` dialog box.	An expression enclosed in quotation marks that evaluates to `true` or `false`, determining which pages are printed or which data is included in a report; default is `null`. You cannot set `printerConditions` to the names of variables.
`printerFieldNames`	Specifies that recordfield names are printed in a report. Setting this property to `true` is the same as checking `Print Field Names As Labels` in the `Print Report Options` dialog box.	`True` if names are printed, or `false`; default is `true`.

PRINTER PROPERTY	DESCRIPTION	VALUE
printerFields	Specifies which recordfields are included in a report. Set this value before setting a value for printerFieldWidths.	A list of recordfield names; default is null.
printerFieldWidths	Specifies the widths of columns in a column report. Set the value for printerFields before setting the value for printerFieldWidths; otherwise, ToolBook may ignore widths that don't have corresponding fields.	A list of numbers in page units. Each number specifies the width of one column. The default is null, which results in balanced widths.
printerGroupsAcross	Specifies the number of groups printed side-by-side in a group report. Setting this property is the same as choosing a Groups Across option in the Print Report dialog box.	A positive integer from 1 to 4; default is 1.
printerGutterHeight	Specifies the vertical distance between pages printed on a sheet, between rows in a column report, or between groups in a group report. This property has no effect when the value of printerArrangement is 1,1.	A number in page units; default is 360 (0.25").
printerGutters	Specifies the vertical and horizontal distances between pages printed on a single sheet, between rows in a column report, or between groups in a group report. This property has no effect when the value of printerArrangement is 1,1.	A list of two numbers in page units, the first representing the value of printerGutterWidth; the second, the value of printerGutterHeight; default is 360,360 (0.25", 0.25").
printerGutterWidth	Specifies the horizontal distance between pages printed on a sheet, between columns in a column report, or between groups in a group report when printerGroupsAcross is greater than 1. This property has no effect when the value of printerArrangement is 1,1.	A number in page units; default is 360 (0.25").
printerLabelWidth	Specifies width of recordfield names for reports when printerFieldNames is true.	A number in page units; default is 2160 (1.5").
printerLeftMargin	Specifies the width of the left margin on the printed sheet.	A number in page units; default is 1440 (1").

PRINTER PROPERTY	DESCRIPTION	VALUE
printerMargins	Specifies values for printerLeftMargin, printerRightMargin, printerTopMargin, and printerBottomMargin.	A list of four numbers in page units that define the left, right, top, and bottom page margins respectively; default for all margins is 1440 (1").
printerName	Specifies the printer to be used. Setting this property changes the default printer listed in the WIN.INI file. We recommend that you do not change the value of this property, because the varying format for printer names between versions of Windows can cause scripts to fail when run under another version of Windows.	A list that contains the name of a printer as it appears in the Print Setup dialog box, the name of a printer driver, and the name of a printer port. If the default printer is an HP LaserJet, the driver is HPPCL.DRV, and the port is LPT1:, the value is typically PCL/HP LaserJet,HPPCL, LPT1:.
printerPageBitmap	Specifies the method for printing pages. Setting this property to true is the same as checking Print As Bitmap in the Print Page Options dialog box.	True if printing at screen resolution, false if printing at printer resolution; default is false.
printerRightMargin	Specifies the width of the right margin on the printed sheet.	A number in page units; default is 1440.
printerScaling	Specifies the size and scale of print margins. Setting this property is the same as choosing a Scale type in the Print Page Options or Print Report Options dialog box.	Actual, printer, or custom; default is custom.
printerSize	Specifies the width and height of groups in a report. Set the value of printerSize, for example, to ensure that data fits a mailing label size. Data outside that size is clipped.	List of two positive numbers in page units. If either number is 0, the size of the group is automatically calculated to accommodate the number of groups, paper size, margins, gutters, and recordfield widths; default is 0,0.
printerStyle	Specifies the style in which data is printed in reports.	Pages, columns, or groups; default is pages.
printerTopMargin	Specifies the width of the top margin on the printed sheet.	A number in page units; default is 1440 (1").

printerFonts()

TBWIN.DLL Font Functions

SYNTAX `printerFonts(<typeface name>)`

DESCRIPTION Queries a list of available sizes for fonts available for printing.

This function will only generate font sizes for True Type fonts, and not for Screen fonts. Refer to `displayFonts()` if you are interested in Screen fonts.

RETURNS If the parameter is `null`, `printerFonts()` returns a string composed of textlines. Each textline is a list containing the `name` of a typeface and the character sizes available for that typeface.

If the parameter is the name of a typeface, `printerFonts()` returns a list containing the typeface `name` and the character sizes available for that typeface.

If the named typeface is not available, it returns only the name of the typeface.

If an error occurs, the returned value is `null` and `sysError` is set to a negative value.

Example:
`request printerFonts("Arial")` returns this string:
`arial,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48,`
`50,52,54,56,58,60,62,64,66,68,70,72`

Even though `Arial` is a True Type font and is not limited to certain font sizes, the sizes reported back is the string above, ranging between 8 and 72 points.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    STRING printerFonts(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<typeface name>`	The name of the typeface for which you want the available sizes and styles. Use `null` for a list of all typefaces.

EXAMPLES

```
-- Generate a list of all printer fonts, but don't show sizes
fnts = printerFonts(null)
step k from 1 to textlineCount(fnts)
    textline k of fnts = item 1 of textline k of fnts
end
text of field "allFontList" = fnts
```

propertyInfo()

Object Function

SYNTAX `propertyInfo(<object reference>,<"property name">)`

DESCRIPTION Returns a list of items that provide information about the specified property of the specified object.

The number of entries in the list may vary, depending on each property's unique requirements.

The position in list determines the following values:

POSITION	SPECIFIES
1	Reserved.
2	If `true`, you can get the property.
3	If `true`, you can set the property.

POSITION	SPECIFIES
4	Reserved.
5	If `true`, the property is directly editable in the `Property editor`.
6	Reserved.
7	Reserved.
8	The data type of the property; for details, *see the following table.
9	Reserved.
10	Reserved.
11	Reserved.
12	Reserved.
13	If `true`, the property is an `array`.

* Values and data types for list item 8

0	Unknown	6	Point	12	Resource
1	Logical	7	Dword	13	Indents
2	String	8	Custom value	14	Short
3	Word	9	Enumerated list	15	Long
4	Stack	10	RGB color	16	Float
5	Rect	11	HLS color	17	Double
18	Script	19	OCX (general)	20	OCX font
21	OCX picture	22	OCX color		

PARAMETERS

PARAMETER	DESCRIPTION
`<object reference>`	A valid reference to an object, which can include a reference to a book.
`<property name>`	The name of a property of the object.

EXAMPLES

```
obj = gauge "turn"
pList = null
propList = propertyList(obj)
step i from 1 to itemCount(propList)
    put propertyInfo(obj,item i of propList) & CRLF after pList
end
text of field "props" = pList
```

propertyList()

Object Function

SYNTAX

```
propertyList(<object reference>)
```

DESCRIPTION

Returns a list of all properties of the specified object.

NOTES

Note that this does not include `User Properties`. If you need a list of all `User Properties` of an object you can get this value by accessing the `userProperties` property of the object.

PARAMETERS

PARAMETER	DESCRIPTION
`<object reference>`	A valid reference to an object, which can include a reference to a book.

EXAMPLES

```
props = propertyList(target)
```

push

Data Manipulation

SYNTAX `push [<expression>] [onto <stack>]`

DESCRIPTION Adds an item or a value to a `stack`, which is a list of items.

If a `stack` is not specified and the item is the `unique name` of a page, the item is pushed onto `sysHistory`.

NOTES The `push` command adds the item to the beginning of the list (the item is pushed onto the top of the `stack`), unless you specify another position.

You cannot push an item other than the `unique name` of a `page` onto `sysHistory`. To see which items are in the system history stack, look at the value of the `sysHistory` property, which contains a list of the `unique names` of the items in the system history.

Items pushed onto a stack can be retrieved using the `pop` command. For example, if you have a page you want the user to return to at the end of each of several sections, instead of using a statement such as `go to page 1`, you can use the statement `push page "Summary"` onto `svBookmarks`, where `svBookmarks` is the name of a system variable used as a `stack`.

When it's time for the user to return to the summary, you can use the statements `pop svBookmarks` and `go to it` to display the page with the summary. That way, `page "Summary"` will still be displayed, even if its page number has changed.

`Pushing` and `popping` items is much faster than building or accessing a `stack` using the `put` and `get` commands.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	Any expression.
`<stack>`	A reference to any list, such as a variable or property name, or textlines in the text of a field; the default is `sysHistory`.

EXAMPLES

```
-- collect a list of named backgrounds in the book
bgs = backgounds of this book
allNamed = null
while bgs <> null
    pop bgs into curBgnd
    if namd of curBgnd <> null
        push curBgnd onto allNamed
    end
end

-- push an item into a specific position of a variable
push "dog" onto item 5 of vAnimals
```

put

Assigning Values

SYNTAX `put <expression> [into | before | after <container>]`

DESCRIPTION Copies the value of an expression into, before, or after the contents of a container.

If `<container>` is not specified, the value of `<expression>` is put into the `Command Window` and ToolBook shows the Command Window. The Command Window does not exist in the `Runtime` version of ToolBook so this usage of put is intended for `Authoring` purposes.

The `put...into` form replaces the contents of `<container>` with the value of `<expression>`.

The `put...before` form places the value of `<expression>` before the existing contents of `<container>`.

The `put...after` form places the value of `<expression>` after the existing contents of `<container>`.

NOTES You can also use the `set` command to assign a value to a container.

If you attempt to use `put` with a property name that is not a standard property, ToolBook will search the object hierarchy for a corresponding `to set` handler for that property name. If no such handler exists, ToolBook then treats the name as a `user property` and sets its value.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	A valid expression.
`<container>`	A reference to a container.

EXAMPLES

```
-- show the current value of x in the Command Window
put "At this point in the code, the value of x is: " & x

-- These two do the same thing
put 5 into x
x = 5

put "Dear, " before text of field "letter"

a = "The "
b = "end!"
put b after a
```

pv()

Financial Values

SYNTAX `pv(<rate>,<nper>,<payment>[,<future value>][,<type>])`

DESCRIPTION A financial function that returns the present value of an investment. The present value is based on periodic, constant payments and a constant interest rate to reach a future value. Use the `<type>` parameter to designate whether the investment is an ordinary annuity or an annuity due.

PARAMETERS

PARAMETER	DESCRIPTION
`<rate>`	A number representing the interest rate per period.
`<nper>`	A positive number representing the total number of payment periods.
`<payment>`	A number representing the payment made each period; it must remain constant over the life of the investment.
`<future value>`	Optional Parameter. A number representing the cash balance after the last payment is made. If `<future value>` is omitted, it is assumed to be `0`.
`<type>`	Optional Parameter. A value of `true` or `false` that indicates when payments are due. `True` indicates that payments are due at the beginning of the period (annuity due), `false` at the end (ordinary annuity). If `<type>` is omitted, it is assumed to be false.

EXAMPLES

```
x = pv(0.095,10,3000,0,true)
```

quote

Special Constants

DESCRIPTION Represents a quotation mark (").

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES

```
-- This works but is hard to read from a programmers perspective
newString = """" & oldString & """"

-- This also works, and is much easier to read
newString = quote & oldString & quote
```

random()

Arithmetic Values

SYNTAX `random(<integer>)`

DESCRIPTION Returns a random integer between `1` and a specified integer, inclusive.

NOTES It is important to set a `seed` value when using the random function. Refer to `seed` for more information.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<integer>`	Any positive integer from `1` to `32767`.

EXAMPLES

```
-- go to a random page in the book
rnd = random(pagecount of this book)
go page rnd
```

rate()

Financial Values

SYNTAX `rate(<nper>,<payment>,<present val>[,<future val>][,<type>],<guess>])`

DESCRIPTION A financial function that returns the interest rate per period for an investment. This function calculates the interest rate required for a current investment to reach its future value based on the number of compounding periods. Enter a negative number for payments out-of-pocket and a positive number for income.

PARAMETERS

PARAMETER	DESCRIPTION
`<nper>`	A positive whole number representing the total number of payment periods.
`<payment>`	A number representing the payment made each period; it must remain constant over the life of the investment.
`<present val>`	A number representing the current value of the investment, or the lump-sum amount that a series of future payments is worth at the present time.
`<future val>`	Optional Parameter. A number representing the future value of the investment, or the cash flow balance to be attained after the last payment is made. If `<future val>` is omitted, it is assumed to be `0`.
`<type>`	Optional Parameter. A value of `true` or `false` that indicates when payments are due. `True` indicates that payments are due at the beginning of the period (annuity due), `false` at the end (ordinary annuity). If `<type>` is omitted, it is assumed to be `false`.
`<guess>`	Optional Parameter. A number between `-0.1` and `0.1` that represents the expected rate of cash flow. If `<guess>` is omitted, it is assumed to be `0.1`.

EXAMPLES

```
-- Calculates the interest rate for a 5-year loan for $10,000
x = rate(60,-200,10000,0,false,0.1)
```

reader

Working Level

DESCRIPTION Sent to the page when you choose `Reader` from the `View` menu.

You can also `send` this message using the `send Reader` statement. ToolBook's default response is to change the `working level` and `sysLevel` property to `Reader`.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle enterPage
    forward
    send reader
end
```

readerStatusBar

Property

DESCRIPTION A property of a viewer that specifies whether the status bar is visible at `Reader` level in the viewer when it is first opened.

NOTES You can get or set this property.

VALUES `True` or `false`.

The default is `false`.

If `false`, the status bar is not visible when the viewer is displayed at `Reader` level.

EXAMPLES

```
readerStatusBar of viewer "help" = true
```

readerVisible

Property

DESCRIPTION A stage property that specifies whether the stage is visible at `Reader` level.

NOTES You can get or set this property.

VALUES `True` or `false`.

The default is `true`.

If `true`, the stage is visible at both `Author` and `Reader` levels.

If false, the stage is not visible at `Reader` level.

EXAMPLES

```
ReaderVisible of stage "player" = false
```

readFile

File Manipulation Command

SYNTAX

```
readFile <file name> to <character>
```

```
readFile <file name> for <number of characters>
```

DESCRIPTION Reads from a specified `ASCII` file, starting at the current file position and continuing until ToolBook encounters the specified character or reads a specified number of characters.

NOTES The file must have been opened with the `openFile` command.

The characters read from the file are put into `IT`. When ToolBook reads to a specified character, that character is not included in the text that ToolBook puts into `IT`.

When a file is first opened, the file position is the first character in the file. As ToolBook reads each character in the file, the file position advances to the next character. Therefore, if you perform additional operations on the file, the operations take place at the position that immediately follows the last character read.

When ToolBook reaches the end of a file, ToolBook sets `IT` to `null` and sets `sysError` to `end of file` (`sysErrorNumber = 565`). If you are reading through a file of unknown length, check `IT` or `sysError` for these values after each execution of the `readFile` command.

After ToolBook reads from a file, you must explicitly close the file with the `closeFile` command if it is to be read again or written to in the same session. Otherwise, the `Execution Suspended` message appears.

To read to the end of a file, use the `EOF` constant with the `readFile <file name> to <character>` syntax. For example, `readFile "file.txt" to EOF` puts the file into `IT`. (ToolBook does not read in the EOF character.) The variable `IT` like any other ToolBook string variable is limited to a capacity of 64K of storage. Be careful not to try to read in a file larger than 64K all at once.

Do not attempt to use the statement while `IT` is not `EOF` with the `readFile` command; doing so results in an infinite loop. The `EOF` constant is `Control-Z`, the DOS standard. The `EOF` constant cannot be used as a condition; ToolBook never actually reads the `EOF` constant into a string with the `readFile` command. When ToolBook reads a file and reaches the `EOF` constant, ToolBook puts `null` into `IT`. ToolBook also puts `null` into `IT` if the actual string is shorter than specified. For example, `readFile for 1000` may return only `100` if there are no more than `100` characters.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	A file name, including the path if necessary.
`<character>`	A literal character or a constant such as `EOF`, `quote`, or `CR`.
`<number of characters>`	An expression that yields a non-negative integer.

EXAMPLES

```
-- Reads to the end-of-file character
readFile "docudram.txt" to EOF
```

```
-- Imports a text file into the text of a field and replaces
-- CRLF characters with spaces
to handle buttonClick
    openFile "import.txt"
    clear text of field "WP text"
    while sysError is not "end of file"
        readFile "import.txt" to LF
        if last char of IT is CR
            clear last char of It --Clears CR character
        end
        put space && IT after text of field "WP text"
    end
    closeFile "import.txt"
end
```

```
-- Imports a file that contains fixed-length records of 80 characters
to handle buttonClick
    system svDataFile
    clear sysError
    openFile svDataFile
    readFile svDataFile for 80
    while IT is not null and sysError is null
        send processRecord IT
        readFile svDataFile for 80
    end
    closeFile svDataFile
end
```

recordfield

Object Type

DESCRIPTION The object type name for a `recordfield`, which is a special type of field that can only be created on a `background`.

NOTES The parent of a recordfield is its background, or a group if the recordfield is in a group.

The recordfield appears on each page that shares its background, and different text can be typed in the recordfield of each page. Users and developers can print reports and export data from recordfields.

REFER TO A RECORDFIELD	USE THIS SYNTAX
On the current background	`recordfield "Name List" of \` `  this background`
In a group	`recordfield "Label" of group \` `  "Records" of this background`
On another background	`recordfield "Phone" of background \` `  "Data"`

TEXT OF A RECORDFIELD	USE THIS SYNTAX
On the current page	`text of recordfield "List"`
On another page	`text of recordfield "List" of page 3`
In another book	`text of recordfield "Item" of page 6 \` `  of book "order.tbk"`

Because a recordfield is on the background, it appears in the same style and position on every page that shares the same background, even though the recordfield can hold different text on each page of that background.

When you refer to the recordfield itself, or any of its properties except text, you refer to it in relation to the background where it is located. When you refer to the text in a recordfield, you must refer to the specific page where that text is entered.

The `enterRecordField` and `leaveRecordField` messages are sent to a recordfield at `Reader` level when it gets or loses the focus. The `destroy`, `make`, `moved`, and `sized` notification messages can be sent to recordfields.

Removing a recordfield deletes all of the text it stores on each page that shares the background. You cannot undo the removal of a recordfield object.

rectangle

Object Type

SYNTAX `draw rectangle from <location> to <location>`

DESCRIPTION The object type name for a `rectangle`.

NOTES A rectangle's parent can be a page, background, or group.

For a rectangle, the `bounds` and `vertices` properties have the same value.

red

Color Constants

DESCRIPTION Represents the color red. This is equivalent to the RGB value of 255,0,0 and the HLS value of 0,50,100.

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES

```
rgbFill of field "list" = red
```

```
if rgbStroke of rectangle "drop area" = red
    rgbStroke of rectangle "drop area" = 0,191,0
end
```

```
strokeColor of ellipse id 14 = red
```

remove menu

Menu Command/Function

SYNTAX

```
remove menu <name | alias> [in <menu reference> [in <menu reference>] ... ]
  [at <level>]
```

DESCRIPTION Removes a specified menu from the menu bar in the target window and shifts other menus to the left.

NOTES You can remove a menu from a submenu using the in <menu reference> parameter. You can use this parameter multiple times to refer to a cascading menu with multiple submenus.

An error will result if a handler attempts to remove a menu that doesn't exist.

Removing menus at Author level generates an error in Runtime ToolBook.

PARAMETERS

PARAMETER	DESCRIPTION
<name | alias>	The name of a menu as it appears on the menu bar, or the alias assigned to the menu.
<menu reference>	A valid reference to an existing menu or submenu in the target window. If more than one menu shares the same reference, ToolBook acts on the first menu from the left on the menu bar.
<level>	Author, Reader, or both. If a level is not specified, the current working level is assumed.

EXAMPLES

```
remove menu "Size" in menu "Options" at Reader
```

remove menuItem

Menu Command/Function

SYNTAX

```
remove menuItem <name | alias> [in menu <name | alias> [in <menu
  reference>] ... ] [at <level>]
```

DESCRIPTION Removes a menu item from the menu bar in the target window.

NOTES You can remove a menu item from a submenu using the in menu <name | alias> in <menu reference> parameter. You can use this parameter multiple times to refer to a cascading menu with multiple submenus.

An error will result if a handler attempts to remove a menu item that doesn't exist.

PARAMETERS

PARAMETER	DESCRIPTION
`<name | alias>`	The name of the menu item as it appears on the menu, or the alias assigned to the menu item.
`<menu reference>`	A valid reference to an existing menu or submenu in the target window. If more than one menu shares the same reference, ToolBook acts on the first menu from the left on the menu bar.
`<level>`	`Author`, `Reader`, or `both`. If a level is not specified, the current working level is assumed.

EXAMPLES

```
remove menuItem "clearAll" at Reader

remove menuItem "Left" in menu "Align" in menu "Draw"
```

remove resource

Resource Commands

SYNTAX

```
remove resource <resource reference>
```

DESCRIPTION

Removes a specified resource from a book.

A resource can only be removed if there are no current references to it in the book. Use `resourceInfo()` to determine if the number of references to a resource is at `0`.

PARAMETERS

PARAMETER	DESCRIPTION
`<resource reference>`	A valid reference to a resource.

EXAMPLES

```
remove resource icon "ball"

remove resource menuBar "main menu" of book "training.tbk"
```

remove separator

Menu Command/Function

SYNTAX

```
remove separator <number> in menu <name | alias>
  [in <menu reference>] ... ] [at <level>]
```

DESCRIPTION

Removes a `separator bar` between menu items from the menu bar in the target window.

NOTES

Removes only one separator bar at a time.

You can remove a separator in a submenu using the in `<menu reference>` parameter. You can use this parameter multiple times to refer to a cascading menu with multiple submenus.

PARAMETERS

PARAMETER	DESCRIPTION
`<number>`	A positive integer that indicates the order of separator bars in the menu. For example, if `<number>` is 1, the first separator bar in the menu is removed.
`<name | alias>`	The name of a menu as it appears on the menu bar, or the alias assigned to the menu.
`<menu reference>`	A valid reference to an existing menu or submenu in the target window. If more than one menu shares the same reference, ToolBook acts on the first menu from the left on the menu bar.
`<level>`	`Author`, `Reader`, or `both`. If a level is not specified, the current working level is assumed.

EXAMPLES

```
--Removes the second separator bar from the menu "file"
remove separator 2 in menu "file" at both
```

remove sysBook

System Book Related Commands

SYNTAX `remove sysBook [<file name>]`

DESCRIPTION Removes a bound system book from a book.

To determine which system books are currently loaded, check the `sysBooks` system variable.

To determine which system books have been `bound` to the current book, call `queryAddedExtension()` and search through the returned stack for system books, or open the `Bound System Books` dialog box.

NOTES A system book can be added in two ways. First is via the bound system books mechanism, and secondly is by adding the system book reference to the `sysBooks` system property.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	An optional parameter that specifies the name of the bound system book to be removed. If `<file name>` is omitted, the default response is to open the `Bound System Books` dialog box.

EXAMPLES

```
remove sysBook "MYSBK.SBK"
```

removeDirectory()

TBDOS.DLL File Function (Directory)

SYNTAX `removeDirectory(<path name>)`

DESCRIPTION Deletes the specified directory. The directory must be empty.

RETURNS

`1` Directory was successfully deleted.
`-3` Specified path was invalid or the directory was not empty.
`-5` Access to the directory was denied.
`-16` Current directory was specified.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    INT removeDirectory(STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `removeDirectory32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<path name>`	The path of the directory to be deleted.

EXAMPLES

```
get removeDirectory("c:\myPersonalProjectFolder")
```

removeDirectory32()

TBFILE32.DLL File Function (Directory)

SYNTAX `removeDirectory32(<path name>)`

DESCRIPTION Deletes the specified directory. The directory must be empty.

RETURNS

`1` Directory was successfully deleted.

Otherwise, the function returns one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    INT removeDirectory32(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<path name>	The path of the directory to be deleted.

EXAMPLES

```
get removeDirectory32("c:\myPersonalProjectFolder")
```

removeFile()

TBDOS.DLL File Function (General)

SYNTAX

```
removeFile(<file name>)
```

DESCRIPTION Deletes the specified file.

RETURNS

```
 1   Specified file was successfully deleted.
-2   Specified file was not found.
-3   Specified path was invalid.
-5   Access to the file was denied.
```

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    INT removeFile(STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `removeFile32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
<file name>	The name and optionally the path of the file to delete.

EXAMPLES

```
to handle buttonClick
    fName = "c:\project\doorslam.wav"
    get removeFile(fName)
end
```

removeFile32()

TBFILE32.DLL File Function (General)

SYNTAX

```
removeFile(<file name>)
```

DESCRIPTION Deletes the specified file.

RETURNS

```
 1   Specified file was successfully deleted.
```

Otherwise, the function returns one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    INT removeFile32(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<file name>	The name and optionally the path of the file to delete.

EXAMPLES

```
to handle buttonClick
    fName = "c:\project\doorslam.wav"
    get removeFile32(fName)
end
```

removeHotword

Hotword Message

DESCRIPTION

Sent to the page when you choose Remove Hotword from the Text menu.

You can also send this message using the send removeHotword statement. ToolBook's default response is to change a hotword in the selected text back to ordinary text, effectively destroying the hotword.

NOTE

As with most all system generated messages, it is important to forward the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle buttonClick
    select third word of text of field "Info"
    send removeHotword
end

to handle removeHotword
    request "Sorry, you are not permitted to remove hotwords on this page."
end
```

replace resource

Resource Commands

SYNTAX

```
replace resource <resource reference> with <file name>

replace resource <resource reference> with resource <resource reference>
```

DESCRIPTION

Replaces the contents of a resource while preserving the resource's name and ID number.

Use the replace resource command to replace one resource with another without having to update all references to the resource in scripts.

PARAMETERS

PARAMETER	DESCRIPTION
<resource reference>	A valid reference to the original resource. You can even refer to a resource in another book.
<file name>	A string that specifies the name of the file to import, including the path if necessary. ToolBook supports the following file formats: .BMP, .DIB, .WMF, .CUR, .GIF, .ICO, JPG, .MNU, .PAL, .TTF. or .TXT. **NOTE** The replace resource...with <file name> command only supports standard Windows bitmap formats (BMP, DIB, and WMF) at runtime, because other graphics filters are not redistributable.

EXAMPLES

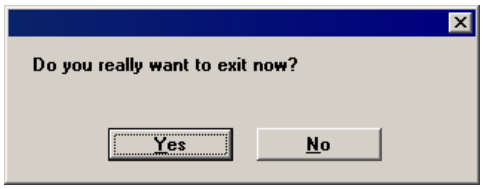
```
replace resource icon "clock" with "c:\tb\hourglass.ico"

replace resource bitmap "x3" with resource bitmap "y1" of book "c:\test.tbk"
```

request

Dialog Box

SYNTAX `request <question> [with <reply> [or <reply2> [or <reply3>]]]`

DESCRIPTION Displays a modal dialog box with a question or an informational message, and up to three buttons, each representing a different reply.



RETURNS The user's response is placed into `IT` when the dialog box is dismissed. If the user clicks a button or presses `Enter`, `IT` is set to the string that is the `caption` of the button that had the focus. If the user presses `Esc` or closes the dialog box using the Control menu, ToolBook sets `IT` to `cancel` and `sysError` to `cancel`.

Even if your `Request` dialog box does not include a `Cancel` button, the user has the option of pressing `Esc`, so your script should check for this.

NOTES This dialog box is very limited in functionality. It is recommended that you use `ASYM_Request()` instead, which has many more features to offer.

If a reply is not specified, ToolBook puts a button labeled `OK` into the dialog box. The user cannot continue to work in the current instance of ToolBook until one of the buttons is clicked or the `Esc` key is pressed.

PARAMETERS

PARAMETER	DESCRIPTION
`<question>`	Any string expression.
`<reply>`	One or more strings. The first `<reply>` appears on the left as the default button. Place an `&` before a character in `<reply>` if you want to define a mnemonic access character (the value of `IT` does not include the ampersand when the user chooses that reply option).

EXAMPLES

```
request "Do you really want to exit now?" with "&Yes" or "&No"
```

reset

Variable Command

SYNTAX `reset <array name>`

DESCRIPTION Resets the dimensions of a dynamic array. Using the `reset` command with a fixed array results in an `Execution Suspended` error.

PARAMETERS

PARAMETER	DESCRIPTION
`<array name>`	The name of a dynamic array variable.

EXAMPLES

```
-- Creates a dynamic array, then resets its.
local myArray[]
step ctr from 1 to 10
    myArray[ctr] = ctr
end
reset myArray
```

resourceCount()

Resource Function

SYNTAX `resourceCount(<type>,<book reference>)`

DESCRIPTION Returns the number of resources of the specified type that exist in the specified book.

PARAMETERS

PARAMETER	DESCRIPTION
`<type>`	A valid resource type name: `bitmap`, `clip`, `cursor`, `font`, `icon`, `menuBar`, `palette`, or `sharedScript`.
`<book reference>`	A valid reference to the book containing the resources to display.

EXAMPLES

```
ct = resourceCount(bitmap,this book)
```

resourceInfo

Resource Property

DESCRIPTION A property of a resource that specifies information about that resource.

The information available varies depending on the type of resource. This property **cannot** be set.

VALUE A list of items that specify information about the resource. The value varies depending on the resource type.

The following table summarizes the items returned by the `resourceInfo` property for each type of resource.

TYPE	ITEMS IN RETURN VALUE
`bitmap`	Reference count, bitmap width in `pixels`, bitmap height in `pixels`, color depth, bitmap size in bytes.
`Clip`	Reference count, name, media type, media source, start point, end point, time format.
`Cursor`	Reference count, icon width in `pixels`, icon height in `pixels`, color depth, hotspot x position, hotspot y position.
`Font`	Reference count, license category, font family, font style, character set. Note that the Reference count for fonts is always 0.
`Icon`	Reference count, icon width in `pixels`, icon height in `pixels`, color depth.
`MenuBar`	Reference count.
`Palette`	Reference count, number of palette entries.
`SharedScript`	Reference count.

EXAMPLES

```
colorDepth = item 4 of resourceInfo of bitmap "star"
```

resourceList()

Resource Function

SYNTAX `resourceList(<type>,<book reference>)`

DESCRIPTION Returns a list of all valid resource references of the specified type that exist in the specified book.

The total number of items in the list indicates the number of resources of that type that exist within the book.

PARAMETERS

PARAMETER	DESCRIPTION
`<type>`	A valid resource type name: `bitmap`, `clip`, `cursor`, `font`, `icon`, `menuBar`, `palette`, or `sharedScript`.
`<book reference>`	A valid reference to the book containing the resources to display.

EXAMPLES

```
allBmps = resourceList(bitmap,this book)
```

resourceUsedBy()

Resource Function

SYNTAX `resourceUsedBy(<resourceRef>,<bookRef>)`

DESCRIPTION Returns a list of the objects using any one resource.

RETURNS A list of objects using a resource, or `null` if no references were found.

PARAMETERS

PARAMETER	DESCRIPTION
`<resourceRef>`	A reference to the resource.
`<bookRef>`	A reference to the book containing the resource.

EXAMPLES

```
allRcs = resourceUsedBy(bitmap "star",this book)
```

restore menubar

Menu Command/Function

SYNTAX `restore menubar [at <level>]`

DESCRIPTION Restores the `menu bar` in the target window to the default setting of the menu bar resource assigned to that window or to the built-in defaults for the ToolBook `Author`-level menu bar.

NOTES When the `restore menuBar` command is executed, any `menus` and `menu items` added with OpenScript commands are removed from the menu bar resource in the target window.

If the at `<level>` parameter is `author`, all user-defined menus and menu items are removed from ToolBook's `Author`-level menu bar, regardless of which viewer is the target window.

PARAMETERS

PARAMETER	DESCRIPTION
`<level>`	`Author`, `Reader`, or `both`. If a level is not specified, the current working level is assumed.

EXAMPLES

```
restore menubar at Reader
```

restore system

Script Control

SYNTAX `restore system`

DESCRIPTION Destroys all system variables, unlinks all `DLLs`, cancels all message translation, and restores most system properties to their default values.

This command restores ToolBook to a known configuration and reclaims memory from system variables you no longer need.

Executing this command unlinks any `DLLs` that are used by your system books. When you execute this command, ToolBook sends the `systemRestored` notification message.

The `restore system` command has no effect on any system printer properties or on the `magnification`, `mousePosition`, `sysHistory`, `sysLevel`, `sysOperatingSystem`, and `sysVersion` properties.

If a system property has a corresponding startup system property, restore system sets the system property to its startup value. For example, restore system sets `sysBooks` to the value of `startupSysBooks`. This command does not reset the values of startup system properties.

NOTES The restore system command should be used only in situations when a complete cleanup of the system is truly needed. Because this command unlinks all DLLs and clears all system variables, the behavior of your application can be severely affected.

In general, Click2learn does not recommend the use of this command.

EXAMPLES

```
to handle buttonClick
    restore system
end
```

return

Program Flow

SYNTAX `return <expression>`

DESCRIPTION Returns the result of an evaluation of a user-defined function.

This command can only be used within a `to get` handler (and a `to get` handler must use the `return` command).

NOTES The `return` command acts as a break for the function. No statements following the `return` command will be executed.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	Any expression.

EXAMPLES

```
to get quoteIT str
    val = QUOTE & str & QUOTE
    return val
end
```

revertFocus

Property

DESCRIPTION A property of a viewer that specifies whether the viewer keeps the focus.

NOTES You can get or set this property.

If `true`, the viewer that receives the focus will revert the focus to the previous viewer when the system returns to an idle state.

Setting the `revertFocus` property to `true` is similar to setting a viewer's enabled property to `false`, with one important difference: when the user of an application clicks objects in a viewer that has revertFocus set to true, the viewer still sends messages and executes scripts.

VALUES `True` or `false`.

The default is `false`.

If `false`, the viewer can receive the focus as normal.

If `true`, the viewer cannot retain the focus at `Reader` level. When the user clicks the viewer, ToolBook runs the appropriate scripts for the viewer. After the scripts finish running, ToolBook reverts the focus to the last viewer that was the focus window.

EXAMPLES

```
revertFocus of viewer "help" = true
```

rgbFill

Property

DESCRIPTION A property of a background, draw object, recordfield, field, stage, button, combobox or graphic object that specifies the object's fill color in `RGB` values.

NOTES This property can be used interchangeably with `fillColor`, but its values are expressed in `RGB` values instead of `HLS` values. When you change the setting for `rgbFill`, the setting for `fillColor` changes as well.

VALUES The stored value will be a list of three non-negative numbers representing the `red`, `green`, and `blue` values of a single color.

When setting this value you can use valid color constant or a list of three non-negative numbers representing the `red`, `green`, and `blue` values of a single color.

The default is the value of `sysRGBFill` when the object was created. The valid range for any of the three values is `0` to `255`.

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES

```
rgbFill of button "Apple" = red
rgbFill of button "Apple" = 255,0,236
rgbFill of button "Apple" = rgbFill of button "Orange"
```

rgbMat

Property

DESCRIPTION A property of a viewer that specifies the color used by ToolBook to fill any exposed sections of the viewer's client area.

NOTES You can get or set this property. The `rgbMat` is visible in areas of the viewer that are not covered by the client window or by the viewer's child windows when the viewer is resized to expose its client area.

VALUES A list of three `RGB` values, or a valid ToolBook `color constant`.

The default is `128,128,128`.

EXAMPLES

```
rgbMat of viewer "help" = red
```

rgbStroke

Property

DESCRIPTION A property of a background, draw object, recordfield, field, button, combobox or graphic object that specifies the object's fill color in RGB values.

NOTES This property can be used interchangeably with strokeColor, but its values are expressed in RGB values instead of HLS values. When you change the setting for rgbStroke, the setting for strokeColor changes as well.

VALUES The stored value will be a list of three non-negative numbers representing the red, green, and blue values of a single color.

When setting this value you can use valid color constant or a list of three non-negative numbers representing the red, green, and blue values of a single color.

The default is the value of sysRGBStroke when the object was created.

The valid range for any of the three values is 0 to 255.

EXAMPLES

```
rgbStroke of button "Apple" = red
rgbStroke of button "Apple" = 255,0,236
rgbStroke of button "Apple" = rgbStroke of button "Orange"
```

RGBtoHLS()

TBWIN.DLL Color Functions

SYNTAX

```
RGBtoHLS(<red>,<green>,<blue>)
```

DESCRIPTION Converts the specified RGB color value into an HLS color value.

RETURNS If no error occurs, RGBtoHLS() returns a string of three values that represent hue, lightness, and saturation values.

Otherwise, the function returns:

-20 Memory allocation error.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING RGBtoHLS(WORD,WORD,WORD)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<red>	The numeric value in the range of 0 - 255.
<green>	The numeric value in the range of 0 - 255.
<blue>	The numeric value in the range of 0 - 255.

If any parameter is out of range, the function assumes a value of 0 for that parameter.

EXAMPLES

```
clr = RGBtoHLS(50,120,240)
```

richText

Property

DESCRIPTION A field or recordfield property that specifies text in rich-text format (`RTF`).

NOTES You can get or set this property.

Use this property to transfer formatted text between fields or recordfields, or to read `RTF` files with OpenScript file access commands.

Below is the richText of a field containing the word Hello.

```
{\rtf1 \ansi \deff0 {\fonttbl {\f0 \fswiss MS Sans Serif;}}{\colortbl \red255
\green255 \blue255 ;\red0 \green0 \blue0 ;}\ql \fi0 \li0 \ri0 \sl0 \tx720 \b0
\i0 \strike0 \f0 \fs16 \ulnone \up0 \dn0 \cf1 \cb0 Hello}
```

VALUES A string of text in rich-text format.

EXAMPLES

```
-- save the richText of a field to a file
rt = richText of field "comments"
fn = "c:\comments.rtf"
createFile fn
writeFile rt to fn
closeFile fn

-- copy the formatted text of one field to another
richText of field "A" = richText of field "B"
```

rightButtonDown

Event Message

SYNTAX `rightButtonDown <location>,<isShift>,<isCtrl>`

DESCRIPTION Sent at `Reader` level to the object containing the cursor in its active area when the right mouse button is pressed.

NOTES If no `handler` exists in the object hierarchy for the `rightButtonDown` message and the `Authoring` engine is the active engine, ToolBook checks the value of the `sysReaderRightClick` property. If `true`, ToolBook displays the right-click menu for the target object.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
`<location>`	A list of two numbers in `page units`, indicating the location of the cursor when the mouse button is pressed.
`<isShift>`	`True` or `false`, indicating whether the `Shift` key is pressed in conjunction with the mouse button.
`<isCtrl>`	`True` or `false`, indicating whether the `Ctrl` key is pressed in conjunction with the mouse button.

EXAMPLES

```
to handle rightButtonDown
    if target is button "More info"
        go to page "history" of book "facts.tbk"
    end
end

to handle rightButtonDown loc, sft, ctrl
    if ctrl = true
        start spooler
            print
        end
    end
end
```

rightButtonDoubleClick

Event Message

SYNTAX `rightButtonDoubleClick <location>,<isShift>,<isCtrl>`

DESCRIPTION Sent at `Reader` level to the object containing the cursor in its active area when the right mouse button is pressed and released twice within the double-click time specified in the Windows Control Panel.

NOTES When the user double-clicks the right mouse button, the first click sends a `rightButtonDown` message and the second click sends a `rightButtonDoubleClick` message. Consequently, an object receives messages in the following order: `rightButtonDown`, `rightButtonUp`, `rightButtonDoubleClick`, `rightButtonUp`.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
`<location>`	A list of two numbers in `page units`, indicating the location of the cursor when the mouse button is pressed.
`<isShift>`	`True` or `false`, indicating whether the `Shift` key is pressed in conjunction with the mouse button.
`<isCtrl>`	`True` or `false`, indicating whether the `Ctrl` key is pressed in conjunction with the mouse button.

EXAMPLES

```
to handle rightButtonDoubleClick
    if target is button "More info"
        go to page "history" of book "facts.tbk"
    end
    forward
end

to handle rightButtonDoubleClick loc, sft, ctrl
    if ctrl = true
        start spooler
            print
        end
    end
    forward
end
```

rightButtonUp

Event Message

SYNTAX `rightButtonUp <location>,<isShift>,<isCtrl>`

DESCRIPTION Sent at `Reader` level when the right mouse button is released.

NOTES This message is sent to the object that contained the cursor in its active area when the right mouse button was pressed, causing the `buttonDown` message to be sent.

ToolBook supplies three parameters that specify the location of the cursor when the mouse button was pressed, and whether the `Shift` or `Ctrl` key is also pressed.

When the user clicks an object, the object receives these messages in the following order: `rightButtonDown`, `rightButtonUp`.

If the cursor is outside the ToolBook Main window when the mouse button is pressed, then moved inside the Main window before the mouse button is released, the `rightButtonUp` message is sent to the current page.

Since ToolBook does not have a `rightButtonClick` message, use `rightButtonUp` instead.

EXTRA NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
<location>	A list of two numbers in `page units`, indicating the location of the cursor when the mouse button is pressed.
<isShift>	`True` or `false`, indicating whether the `Shift` key is pressed in conjunction with the mouse button.
<isCtrl>	`True` or `false`, indicating whether the `Ctrl` key is pressed in conjunction with the mouse button.

EXAMPLES

```
to handle rightButtonUp
    if target is button "More info"
        go to page "history" of book "facts.tbk"
    end
    forward
end
```

```
to handle rightButtonUp loc, sft, ctrl
    if ctrl = true
        start spooler
            print
        end
    end
    forward
end
```

rightQuote

Special Constants

DESCRIPTION Represents the `ANSI` value of `187`.

Using a standard font, that character would look like this: »

You might expect it to look like a quotation mark of some sort but it is instead a right chevron character.

EXAMPLES

```
x = leftQuote & "Apple" & rightQuote
```

rightString()

Nonexistent String Functions

SYNTAX `rightString(<string>,<num chars>)`

DESCRIPTION Extracts a number of characters from the right (end) of a string.

RETURNS Returns the last `<num chars>` characters of the string.

NOTES Although this function does not exist in ToolBook you can add it by putting this script in the script of your book at which point you can call this function from anywhere in that book.

```
to get rightString str, num
    ct = charCount(str)
    return chars (ct - num + 1) to ct of str
end
```

Also see `leftString()` and `midString()`.

PARAMETERS

PARAMETER	DESCRIPTION
<string>	An expression that results in a string value.
<num chars>	An expression that results in a whole number.

EXAMPLES

```
-- Last 6 chars of a part number is the bin location
to handle buttonClick
    partNum = textline 44 of field "allParts"
    binNumber = rightString(partNum,6)
    put binNumber into text of field "bin"
end
```

rotateLeft

Orientation Message

SYNTAX

```
rotateLeft [<target>]
```

DESCRIPTION

Sent to the page when you choose `Left` from the `Rotate` submenu on the `Draw` menu.

You can also `send` this message using the `send rotateLeft` statement. ToolBook's default response is to `rotate` the selected object or objects to the `left` by `90` degrees or, if there is no selection, to do nothing.

NOTE

As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
<target>	An optional parameter that specifies the object or objects to be manipulated.

EXAMPLES

```
to handle buttonClick
    send rotateLeft irregularPolygon "bolt"
end

to handle rotateLeft
    request "You must first be in Edit mode to rotate bolts."
end
```

rotateRight

Orientation Message

SYNTAX

```
rotateRight [<target>]
```

DESCRIPTION

Sent to the page when you choose `Right` from the `Rotate` submenu on the `Draw` menu.

You can also `send` this message using the `send rotateRight` statement. ToolBook's default response is to `rotate` the selected object or objects to the right by `90` degrees or, if there is no selection, to do nothing.

NOTE

As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
<target>	An optional parameter that specifies the object or objects to be manipulated.

EXAMPLES

```
to handle buttonClick
    send rotateRight irregularPolygon "bolt"
end

to handle rotateRight
    request "You must first be in Edit mode to rotate bolts."
end
```

round()

Arithmetic Values

SYNTAX `round(<number>[,<exponent>])`

DESCRIPTION Returns a number rounded to the nearest whole number. This is done by rounding fractions of `0.5` or greater up to the next highest whole number.

ACTIONS EDITOR The Actions Editor also supports this feature, but does not support the `<exponent>` parameter.

PARAMETERS

PARAMETER	DESCRIPTION
`<number>`	An expression that results in a number.
`<exponent>`	Optional Parameter - An expression that yields an exponent specifying the precision for the rounding operation. The default is `0`. A positive exponent reduces the precision by rounding off the number at a specified number of digits to the left of the decimal point. A negative exponent increases the precision by rounding off the number at a specified number of digits to the right of the decimal point.

EXAMPLES

```
x = 1234.56
val = round(x)    -- results in 1235

x = 1234.56
val = round(x,-1) -- results in 1234.6

x = 1234.56
val = round(x,1)  -- results in 1230
```

roundedCorners

Property

DESCRIPTION A stage property that specifies whether its outline has rounded corners.

NOTES You can get or set this property.

VALUES `True` or `false`.

The default is `false`.

If `false`, the outline is drawn as a regular rectangle.

EXAMPLES

```
roundedCorners of stage "player" = false
```

roundedRectangle

Object Type

SYNTAX `draw roundedRectangle from <location> to <location>`

DESCRIPTION The object type name for a rectangle with rounded corners.

NOTES A rounded rectangle's parent can be a page, background, or group.

For a rounded rectangle, the `bounds` and `vertices` properties have the same value.

rulers

Property

DESCRIPTION A property of a viewer that specifies whether `rulers` are displayed in the viewer.

NOTES You can get or set this property. This works both at `Reader` level and `Author` level.

VALUES `True` or `false`. The default is `false`.

EXAMPLES

```
rulers of viewer "design" = true
```

run

File Command

SYNTAX

```
run <application> | <file name> [minimized]
```

DESCRIPTION Launches a specified application or opens a specified file with the appropriate application. If you include `minimized`, the application or file is `minimized`.

If the command to run another instance of ToolBook is executed from the `Command window`, the focus remains in the original instance.

If the command is executed from a `script`, the focus is in the new instance unless any subsequent statements in the handler explicitly return control to the original instance.

ToolBook launches the specified application and immediately continues executing OpenScript statements, without waiting for the launched program to become fully operational.

PARAMETERS

PARAMETER	DESCRIPTION
`<application>`	The name of a Windows application, including its path if the application is not in the user's path; this parameter can also optionally include the name of a file and other parameters to be passed to the application.
`<file name>`	The name of a file that runs with the specified application. If you don't use an extension with the file name, ToolBook tries to run the file as a ToolBook book. If the file extension has been registered in Windows, it is not necessary to specify the application name since Windows will find it for you. The path must be included with the application or file name if the path has not already been specified with the DOS PATH command and the application or file is not in the current directory. If any file names used in `<file name>` have a path or file name that include the space character, you must enclose the file name within quotation marks, for example: `quote & "C:\Program Files\xyz\file.exe" & quote`
`<minimized>`	An optional parameter specifying that the application is minimized.

EXAMPLES

```
-- The following examples work with ToolBook (INSTRUCTOR.EXE)
-- The next three examples start calc.tbk if it is in the path
run "INSTRUCTOR.EXE calc.tbk"
run "calc.tbk"
run "calc"

-- Runs and minimizes the Windows clock
run "c:\windows\clock.exe" minimized

-- Runs Windows Write (if installed) with the file mynotes.wri
run "mynotes.wri"

-- Runs Windows Notepad with the file WIN.INI
run "notepad.exe win.ini"

-- Runs My Application.exe if it is in the path and passes
-- My Data File.dat as a parameter
appPath = quote & "My Application.exe" & quote
parmPath = quote & "My Data File.dat" & quote
run appPath && parmPath
```

save

File Message

DESCRIPTION Sent to the page when `Save` is chosen from the `File` menu.

You can also `send` this message using the `send save` statement. ToolBook's default response is to save a book under its existing name. If the book has not yet been named, ToolBook displays the `Save As` dialog box so the user can specify a file name for the book.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle buttonClick
    send save
end
```

```
to handle save
    request "Sorry, you are not permitted to save this book."
end
```

save as

File Command

SYNTAX `save as <file name>,<overwrite>`

DESCRIPTION Saves the current book as a new file with a specified file name.

NOTES If the file specified in `<file name>` is a currently running book, an `Execution Suspended` error occurs.

If the specified file is a `read-only` file, `sysError` is set to `read only` and the `save as` command is not executed.

If the `<overwrite>` parameter is `false` and a file with the specified name already exists, `sysError` is set to `file exists`. Otherwise, the contents of the current book replace the contents of any existing file with the specified name.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	A file name, including the path if necessary.
`<overwrite>`	`True` or `false`, indicating whether an existing file should be overwritten.

EXAMPLES

```
save as "c:\mybooks\oldbook.tbk",true
```

save as EXE

File Command

SYNTAX `save as EXE <file name>,<overwrite>`

DESCRIPTION Saves the current book as an executable file with a specified file name.

NOTES Saving a file with this command builds a standard Windows executable file from the book. The file can then be run when its icon is double-clicked in the Windows Explorer window, provided the necessary Runtime ToolBook files are present.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	A file name, including the path if necessary.
`<overwrite>`	`True` or `false`, indicating whether an existing file should be overwritten.

EXAMPLES

```
save as EXE "features.tbk",true
```

save changes

File Command

SYNTAX `save changes to <book reference>`

DESCRIPTION Saves any changes made to a book.

NOTES The book does not have to be open, but it must be named.

This message does not work on an untitled book.

If `sysChangesDB` is `true` and you execute a command that changes a property in another book, ToolBook asks whether the changes should be saved. You can prevent ToolBook from displaying the Save Changes dialog box by executing the `save changes to <book reference>` statement after changing the property, but before exiting the handler.

PARAMETERS

PARAMETER	DESCRIPTION
`<book reference>`	A valid reference to a book.

EXAMPLES

```
-- Saves changes to the current book
save changes to this book
```

```
-- Changes the text property of a field in another book
text of field "OutOfDate" of page 1 of book "mail.tbk" = true
save changes to book "mail.tbk"
```

saveAs

File Message

DESCRIPTION Sent to the page when `Save As` is chosen from the `File` menu.

You can also `send` this message using the `send saveAs` statement. ToolBook's default response is to open the `Save As` dialog box.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle buttonClick
    send saveAs
end
```

```
to handle saveAs
    request "Sorry, you are not permitted to save this book."
end
```

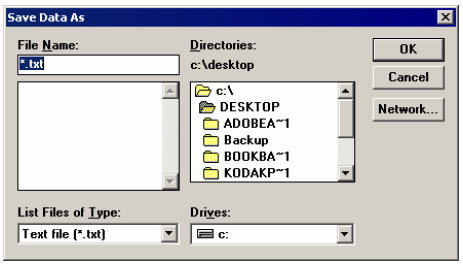
SYNTAX

```
saveAsDlg(<caption text>,<default file>,<default path>,<filters>,
  <index>)
```

DESCRIPTION

Displays the standard Windows short filename Save As dialog box.

This function will accept a long file name as input, but will return short file names only. Use `saveAsDlgLFN()` if you need to return long file names.



RETURNS

If no error occurs, the function returns a string that contains a file name.

Otherwise, the function returns `null` and sets `sysError` to one of these values:

`-1` User canceled the dialog box.
`-2` Memory allocation error.

NOTES

You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING saveAsDlg(STRING,STRING,STRING,STRING,INT)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `saveAsDlg32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<caption text>`	The text displayed in the title bar of the dialog box.
`<default file>`	The default file name. If `null`, all files matching the specified filter are shown.
`<default path>`	The default path. If `null`, the current directory is used.
`<filters>`	A list of file name filters. Each filter is a two-item list containing a description and a wildcard, for example "`Icon (*.ico), *.ico`". This parameter can contain multiple pairs of filters.
`<index>`	An index into the `<filters>` list specifying which to use as the default. The first filter is `1`. To get the filter value, use the `getSaveAsDlgFilterIndex()` function.

EXAMPLES

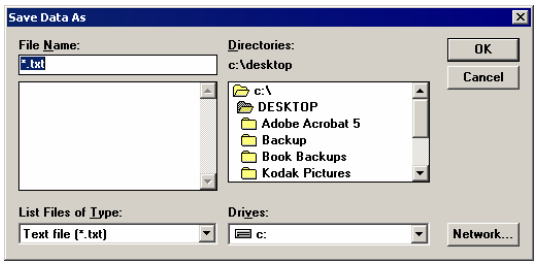
```
to handle buttonClick
    filterList = "Text file (*.txt),*.txt,ToolBook (*.tbk),*.tbk"
    get saveAsDlg("Save Data As",null,null,filterList,1)
    if IT is null
        request "User Canceled."
    else
        request "File selected:" && IT
    end
end
```

SYNTAX

```
saveAsDlg32(<caption text>,<default file>,<default path>, <filters>,
  <index>)
```

DESCRIPTION

Displays the standard Windows Save As dialog box.



RETURNS

If no error occurs, the function returns a string that contains a file name.

Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES

As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    STRING saveAsDlg32(STRING,STRING,STRING,STRING,INT)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<caption text>`	The text displayed in the title bar of the dialog box.
`<default file>`	The default file name. If `null`, all files matching the specified filter are shown.
`<default path>`	The default path. If `null`, the current directory is used.
`<filters>`	A list of file name filters. Each filter is a two-item list containing a description and a wildcard, for example "`Icon (*.ico), *.ico`". This parameter can contain multiple pairs of filters.
`<index>`	An index into the `<filters>` list specifying which to use as the default. The first filter is `1`. To get the filter value, use the `getSaveAsDlgFilterIndex32()` function.

EXAMPLES

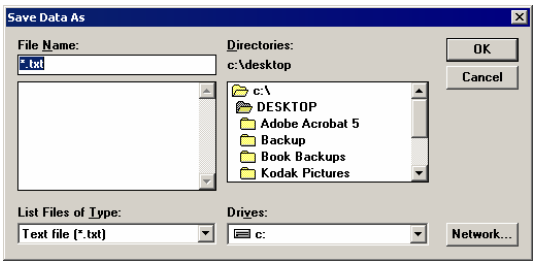
```
to handle buttonClick
    filterList = "Text file (*.txt),*.txt,ToolBook (*.tbk),*.tbk"
    get saveAsDlg32("Save Data As",null,null,filterList,1)
    if IT is null
        request "User Canceled."
    else
        request "File selected:" && IT
    end
end
```

SYNTAX

```
saveAsDlg(<caption text>,<default file>,<default path>,<filters>,
  <index>)
```

DESCRIPTION

Displays the standard Windows Save As dialog box and returns the long file name path.



RETURNS

If no error occurs, the function returns a string that contains a file name.

Otherwise, the function returns `null` and sets `sysError` to one of these values:

`-1` User canceled the dialog box.
`-2` Memory allocation error.

NOTES

You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING saveAsDlgLFN(STRING,STRING,STRING,STRING,INT)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `saveAsDlg32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<caption text>`	The text displayed in the title bar of the dialog box.
`<default file>`	The default file name. If `null`, all files matching the specified filter are shown.
`<default path>`	The default path. If `null`, the current directory is used.
`<filters>`	A list of file name filters. Each filter is a two-item list containing a description and a wildcard, for example "`Icon (*.ico), *.ico`". This parameter can contain multiple pairs of filters.
`<index>`	An index into the `<filters>` list specifying which to use as the default. The first filter is `1`. To get the filter value, use the `getSaveAsDlgFilterIndex()` function.

EXAMPLES

```
to handle buttonClick
    filterList = "Text file (*.txt),*.txt,ToolBook (*.tbk),*.tbk"
    get saveAsDlgLFN("Save Data As",null,null,filterList,1)
    if IT is null
        request "User Canceled."
    else
        request "File selected:" && IT
    end
end
```

saveAsEXE

File Message

DESCRIPTION Sent to the page when you choose `Save As EXE` from the `File` menu.

ToolBook's default response is to open the `Save As EXE` dialog box.

Saving a file with the `EXE` option builds a standard Windows executable file from the book. The file can then be run when the name of the file is double-clicked in Windows Explorer, provided the necessary run-time ToolBook files are present.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle buttonClick
    send saveAsEXE
end

to handle saveAsEXE
    request "Sorry, you are not permitted to save this book."
end
```

saveOnClose

Property

DESCRIPTION A book property that specifies how and when changes are saved when the user exits the book.

NOTES You can get or set this property.

VALUES `Yes`, `no`, `ask`, or `system`.

The default is `system`.

VALUES	DESCRIPTION
`yes`	Changes are saved when the book is exited.
`no`	Changes are not saved when the book is exited.
`ask`	The user is prompted to specify whether changes should be saved.
`system`	The behavior specified by the `sysChangesDB` property is used.

EXAMPLES

```
saveOnClose of this book = "no"
```

screenFromClient()

TBWIN.DLL Screen Display Function

SYNTAX `screenFromClient(<windowHandle>,<point|rectangle>)`

DESCRIPTION Converts a `point` or `rectangle` in `pixels` relative to the upper-left corner of the ToolBook `client area` into a `point` or `rectangle` in `pixels` relative to the upper-left corner of the `screen`.

In the ToolBook Main window, the client area is the working area below the menu bar, not including the scroll bars.

RETURNS If no error occurs, returns the coordinates of either a point or a rectangle, depending on the last parameter.

If an error occurs, the function returns `null` and `sysError` is set to one of these values:

`-20` Memory allocation error.
`-30` `<windowHandle>` is invalid.
`-99` `<point|rectangle>` is invalid.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    STRING screenFromClient(WORD,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<windowHandle>	The value of windowHandle for a viewer such as the mainWindow, this window, or a valid viewer reference.
<point|rectangle>	For a point, a list containing two numbers (x and y coordinates). For a rectangle, a list containing four numbers (coordinates of upper-left and lower-right corners of the rectangle).

screenFromPage()

TBWIN.DLL Screen Display Function

SYNTAX screenFromPage(<windowHandle>,<pageScroll>,<magnification>,<point|rectangle>)

DESCRIPTION Converts a set of coordinates in ToolBook page units to coordinates in pixels relative to the upper-left corner of the screen.

RETURNS If no error occurs, returns the coordinates of either a point or a rectangle, depending on the last parameter.

If an error occurs, the function returns null and sysError is set to one of these values:

-20 Memory allocation error.
-30 <windowHandle> was invalid.
-99 <point|rectangle> was invalid.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    STRING screenFromPage(WORD,STRING,INT,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<windowHandle>	The value of windowHandle for a viewer such as the mainWindow, this window, or a valid viewer reference.
<pageScroll>	The value of pageScroll of mainWindow, this window, or a valid viewer reference.
<magnification>	The value of magnification of mainWindow, this window, or a valid viewer reference.
<point|rectangle>	For a point, a list containing two numbers (x and y coordinates). For a rectangle, a list containing four numbers (coordinates of upper-left and lower-right corners of the rectangle).

EXAMPLES

```
to handle buttonClick loc, isShift, isCtrl
    defaultPosition of viewer "help" = screenFromPage(windowHandle \
        of mainWindow,pageScroll of mainWindow,magnification of     \
        mainWindow,loc)
    show viewer "help"
end
```

screenToClient()

Screen Display Function

SYNTAX `screenToClient(<coordinates>,<viewer reference>)`

DESCRIPTION Converts `pixels` relative to the screen into `pixels` relative to the origin of the `client window` of the specified viewer.

NOTES Use this function to align a viewer that is a child of the viewer's client window with a pop up window. For example, a non-tiled child window is a child of a client window.

PARAMETERS

PARAMETER	DESCRIPTION
`<coordinates>`	A list of numbers in `pixels` that specify a position or positions on the screen. You can include as many numbers as necessary, but they must be in pairs. You can refer to the `bounds`, `position`, `size`, or `vertices` property as the value for `<coordinates>`.
`<viewer reference>`	A valid viewer reference. If the specified viewer is not open, ToolBook displays an error message.

screenToFrame()

Screen Display Function

SYNTAX `screenToFrame(<coordinates>,<viewer reference>)`

DESCRIPTION Converts `pixels` relative to the screen to `pixels` relative to the `client area` of the specified viewer's frame window.

NOTES Use this function to align a viewer that is a child of the viewer's frame window with a popup window. For example, a tiled child window is a child of a frame window.

PARAMETERS

PARAMETER	DESCRIPTION
`<coordinates>`	A list of numbers in `pixels` that specify a position or positions on the screen. You can include as many numbers as necessary, but they must be in pairs. You can refer to the `bounds`, `position`, `size`, or `vertices` property as the value for `<coordinates>`.
`<viewer reference>`	A valid viewer reference. If the specified viewer is not open, ToolBook displays an error message.

screenToPageUnits()

Screen Display Function

SYNTAX `screenToPageUnits(<coordinates>,<viewer reference>)`

DESCRIPTION Converts `pixels` relative to the screen to `page units` for the specified viewer.

Use this function to align an object on the page displayed in the viewer with a popup window. This function accounts for the current values of the viewer's `pageScroll` and `magnification` properties.

NOTES The conversion of screen-relative `pixels` to `page units` is dependent upon the currently installed video driver.

PARAMETERS

PARAMETER	DESCRIPTION
`<coordinates>`	A list of numbers in `pixels` that specify a position or positions on the screen. You can include as many numbers as necessary, but they must be in pairs. You can refer to the `bounds`, `position`, `size`, or `vertices` property as the value for `<coordinates>`.
`<viewer reference>`	A valid viewer reference. If the specified viewer is not open, ToolBook displays an error message.

script

Property

DESCRIPTION A property of all ToolBook object types that is used to hold any OpenScript code that you have written for that object.

NOTES You can get or set this property. Note that you can even set this property at `Runtime`.

VALUES The script of the object.

If `null` the object has no script. If you want to clear the script of an object you can set the script of the object to `null`.

When setting the script of an object, the script has to be syntactically correct. If it is not, a compiling error will occur and ToolBook will generate an `Execution Suspend` error.

EXAMPLES

```
-- create a new button and copy the script from button A to button B
draw button from 0,0 to 300,200
name of selection = "b"
script of button B = script of button A
```

scroll

Property

DESCRIPTION A field or recordfield property that specifies the number of lines that are hidden above the top of a field or recordfield.

NOTES You can get or set this property.

VALUES A non-negative whole number indicating the number of lines that are scrolled above the topmost visible line of the field's displayed text. When the first line of text appears at the top of the field, the value is 0.

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES

```
-- Scroll the text in the field automatically
obj = field "master list"
step k from 1 to textlineCount(text of obj)
    scroll of obj = k
    pause 50
end
```

scrollable

Property

DESCRIPTION A combobox property that specifies whether a combobox displays a scroll bar.

NOTES You can get or set this property.

VALUES `True` or `false`.

The default is `false`.

second

Ordinal Numbers

DESCRIPTION A value used to indicate the second item in a sequence or list. Also used to indicate relative position of pages or backgrounds.

EXAMPLES

```
-- The following paired examples show how you can use ordinal
-- references to make OpenScript easier to read. As you can see it is
-- possible to write your scripts with or without ordinal references.
go to the second page of this book
go to page 2 of this book

if second character of str = "X"
if character 2 of str = "X"

second word of text of field "lesson" = "Hello"
word 2 of text of field "lesson" = "Hello"

put "Cancelled:" into second character of name of button id 16
put "Cancelled:" into character 2 of name of button id 16
```

seed

Data Manipulation

SYNTAX

```
seed <number>
```

DESCRIPTION Sets the starting point for the algorithm used by the `random()` function.

A variable seed, such as the value of `sysTime`, generates various pseudo-random sequences. A constant seed, or specifying no seed, always generates the same pseudo-random sequence each time the application is run.

NOTES You only need to set the seed for a random number generator once in an application, on `enterBook` or `enterApplication`, for example. Setting the seed before each call to random is a common error that may actually reduce the sequence's approach to randomness.

PARAMETERS

PARAMETER	DESCRIPTION
`<number>`	An expression that is a positive integer in the range 0 to 32,767.

EXAMPLES

```
-- standard way to set a seed value
to handle enterBook
    --Initializes the random number generator
    seedNum = sysTime
    format time seedNum as "seconds"
    get seedNum mod 32768
    seed IT
    forward
end
```

seekFile

File Manipulation Command

SYNTAX

```
seekFile <file name> for <position> from <location>
```

DESCRIPTION

Moves the read/write file pointer within an open file.

NOTES

Use this command to gain random access to a file's contents by moving the read/write file pointer to any location in a previously opened file, prior to a read or write operation.

The `readFile` and `writeFile` commands execute from the current position of the file pointer. When the file is first opened, the pointer is positioned to read from the beginning of the file and write to the end of the file. When you first use `seekFile` from the current position, it is from the beginning of the file (read), not the end (write). You can use the `seekFile` command to set the read and write positions of the pointer to the same location.

When `seekFile` executes, ToolBook puts the current file position into `IT`.

For example: `seekFile "fname.txt" for 0 from current` will put the resulting file position into `IT`.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	A file name, including the path if necessary.
`<position>`	A signed number between `-2 billion` and `2 billion` representing the number of bytes and specifying the offset in characters from the specified location. A positive offset moves the pointer toward the end of the file; a negative offset moves the pointer toward the beginning of the file.
`<location>`	`Beginning`, `current`, or `end`, representing the beginning of the file, current position of the file pointer, or end of the file, respectively.

EXAMPLES

```
-- Sets the read/write pointer for the 256 character
seekFile "output.txt" for 256 from beginning

-- Reads the last 500 characters of file
seekFile "output.txt" for -500 from end
readFile "output.txt" to EOF

-- Puts the current file position into IT
seekFile "output.txt" for 0 from current
```

select

Selection Commands

SYNTAX

```
select <objects>
select <string specifier>
select all [<object type>]
select all from <location> to <location>
```

DESCRIPTION

Changes or extends the current selection to include specified objects in the target window.

NOTES Use the `select <objects>` form to select individual objects.

You can select objects on a background only if you are currently on the background layer. Use `send background` to get to the background layer.

You can select objects on a page only if you are currently on the foreground layer. Use `send foreground` to get to the foreground layer.

You cannot select objects on the `foreground` and `background` layer at the same time.

Use the `select <string specifier>` form to select text in a field or recordfield. The text that is selected becomes the value of `selectedText`.

Use the `select all` form to select all objects or all objects of a specific type on either a page or background.

To perform a net selection from one location to another location on the page, use the `select all from...` form.

To `deselect` objects, set the `selection` property to `null` or use the `unselect` command. To extend a selection, use the `extend select` command.

PARAMETERS

PARAMETER	DESCRIPTION
`<objects>`	A list of one or more objects. You cannot select a `book`, `background`, `hotword`, or `viewer`.
`<string specifier>`	A string in a field or recordfield.
`<object type>`	A list of one or more objects. You cannot select a `book`, `background`, `hotword`, or `viewer`.
`<location>`	A list of two numbers in `page units`, specifying the distance from the left edge and top of the page, respectively. The two points defined by `<location>` to `<location>` specify the top left and bottom right corners of a bounding rectangle. Any object completely inside this rectangle is included in the selection.

EXAMPLES

```
to handle specialCut
    focus = null
    select rectangle "box", ellipse "circle"
    send cut
end

--Selects the text of field "names"
select text of field "names"

--Selects a word in a recordfield
select word 1 of textline 3 of text of recordfield "Address"
```

selectChange

Notification Message

SYNTAX `selectChange <text>`

DESCRIPTION Sent to a `combobox` when the user selects a new item from its drop-down list at `Reader` level.

The `text` of the selected item accompanies the message as a `parameter`.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
<text>	A string consisting of the text of the selected item from the drop-down list.

EXAMPLES

```
to handle selectChange itemString
    if itemString = "Car"
        go to page "car maintenance"
    end
    forward
end
```

selectedHotwords

Property

DESCRIPTION A property of a viewer that specifies a list of all hotwords within the currently selected text in the viewer.

NOTES You cannot set this property.

VALUES A list containing the unique name of each selected hotword.

If there is no selection or the text contains no hotwords, the value is null.

Hotwords that are only partially within the selected text are included in the list.

EXAMPLES

```
-- how many hotwords are in the selection
hwds = selectedHotwords of viewer "help"
```

selectedItem

Property

DESCRIPTION A combobox property that specifies which item is selected in the dropdown list box.

NOTES You can get or set this property.

VALUES A positive whole number that represents the item's textline position in the dropdown list.

EXAMPLES

```
selectedItem of combobox "partsList" = 3
```

selectedItemText

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of a combobox, list box or radio button group specifying the text (or caption) of the current selection.

NOTES You can get but not set this p roperty in the Actions Editor. If you need to change the text, see the entry for itemText.

This property has no real meaning to OpenScript and should be used only within the Actions Editor.

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

EXAMPLES

```
On click...
  Display alert: selectedItemText of field "groceries"
```

selectedText

Property

DESCRIPTION A property of a viewer that specifies the currently selected text in the viewer.

NOTES You can get or set this property. If you need to know what text is currently selected, use this `selectedText` property. If you need to select text, use the `select` statement to select the text you are interested in. For example: `select chars 1 to 15 of text of field "information"`.

If text is selected and you set the value of `selectedText`, ToolBook replaces the currently selected text with the new value.

VALUES A string containing the currently selected text.

If there is no selection, the value is `null`.

EXAMPLES

```
txt = selectedText of viewer "help"
```

selectedTextlines

Property

DESCRIPTION A field or recordfield property that specifies which lines of text are selected in a `singleSelect` or `multiSelect` list box field.

NOTES You can get or set this property.

For `multiSelect` list boxes, the last line of text selected has the `focus`.

VALUES A comma separated list of textline numbers in ascending order.

If no textlines are selected the value is `null`.

EXAMPLES

```
-- pre-select some items in this field.
selectedTextlines of field "master list" = "1,2,4,5"

-- does their selection contain at least the 4th one?
if ASYM_ItemInList("4",selectedTextlines of field "list")
    Request "Your selection included the 4th option, great!"
end

-- Once single selection is made determine text of selection
txt = textline (selectedTextlines of field "list") of field "list"
```

selectedTextState

Property

DESCRIPTION A property of a viewer that specifies information about the currently selected text in the viewer. The information includes the location of the selected text relative to the other text in the field, and whether or not the selected text includes hotwords.

NOTES This property cannot be set.

VALUES If no text is selected the value is `null`.

If text is selected, the value is a commas separated list of five items:

VALUE	DESCRIPTION
Item 1	Whole number representing position of first character of selected text.
Item 2	Whole number representing position of last character of selected text.
Item 3	Whole number representing position of first textline of selected text.
Item 4	Whole number representing position of last textline of selected text.
Item 5	`True` or `false`, indicating whether the selected text includes hotwords.

For list box fields, item 3 of selectedTextState is the line number of the line of text that has the focus.

EXAMPLES

```
-- Place these handlers in a book script to preserve the selection
-- on each page as the user navigates through the book
-- (STS and Foc are user properties)
to handle leaveField
    STS of this page = selectedTextState of viewer ID 1
    Foc of this page = focus of viewer ID 1
end

to handle enterField
    select chars (item 1 of STS of this page) \
        to (item 2 of STS of this page) of text of Foc of this page
end
```

selection

Property

DESCRIPTION A property of a viewer that specifies the currently selected object or objects in a viewer.

NOTES You can get or set this property.

When a new object is created, this property is set to the `uniqueName` of the new object.

The selection property can only be set to the current page or to objects on the current page, or, if on the background, to objects on the background.

Viewers, hotwords, backgrounds, and books cannot be selected.

To deselect objects in a script statement, use the `unselect` command or the statement `selection = null`.

VALUES The `uniqueName` of the currently selected object or a list of the unique names of the selected objects.

If there is no selection, the value is `null`.

EXAMPLES

```
selection = objects of group "drawing"
```

```
draw ellipse from 0,0 to 100,50
name of selection = "theGreatEllipse"
position of ellipse "theGreatEllipse" = position of arc "x3"
```

selectionChanged

Notification Message

DESCRIPTION Sent to the page at `Author` level each time the object selection changes.

Any change made to the selection, including clearing the selection, causes the `selectionChanged` message to be sent.

If an object is selected and you click a different object, ToolBook sends this message once.

Get the value of the `selection` property to determine the current selection in the target window.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle selectionChanged
    caption of statusBar = itemCount(selection) && "items selected"
    foward
end
```

self

System Property

DESCRIPTION A system property that specifies the unique name of the object with the currently executing script.

NOTES This property cannot be set.

There are two coding styles when referring to the object with the currently executing script. The first is to use the `self` property and the other is to use the `my` special term. Both work equally well but programmers tend to adopt the use of one or the other.

For the script of a group, it is important to distinguish between `self` and `target`. When the `self` property is used in the script of a group, it refers to the group itself. The `target` property refers to the single object within the group that is the initial recipient of the current message.

When you execute a statement in the Command window, the value of `self` is the `uniqueName` of the currently displayed page.

VALUES The uniqueName of the object with the currently executing script.

EXAMPLES

```
-- both of these effectively do the same thing
caption of self = "hello"
my caption = "hello"
```

sendKeys()

TBWIN.DLL Keyboard Function

SYNTAX

```
sendKeys(<key string>,<return>)
```

DESCRIPTION Sends the specified keys strokes to the active application, processing the keys as if they were typed from the keyboard.

RETURNS If no error occurs, `sendKeys()` returns `0`.

Otherwise, the function returns one of these values:

- `-1` Close brace was missing in the input string.
- `-2` Invalid key name was found in the input string.
- `-3` Close parenthesis was missing in the input string.
- `-4` Invalid repeat count of `0` was specified.
- `-5` The size of the input string was too long, or too many wait parameters were specified.

NOTES Use the optional `{wait <milliseconds>}` clause within the `<key string>` parameter to force the function to pause while sending keys and create the effect of highlighting menus or animating keystrokes.

Use great care when applying `sendKeys()` to operate on other applications. ToolBook has no way to detect or correct errors generated by the other application and always sends the programmed series of keystrokes.

Use the `sendKeys()` function to operate on other applications only when there is no alternative, and then use it with caution. In general, using dynamic data exchange (`DDE`) is a better way for ToolBook to interact with other programs because `DDE` provides a channel for two-way communication between applications, and also provides a path for detecting and dealing with errors in the other application.

As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT sendKeys(STRING,INT)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<key string>	A string of one or more key names or valid key constants, for example: a for the letter a, or {keyEnter} for the Enter key. Use OpenScript key constants within braces {}. You can add a {wait <milliseconds>} clause to the key string to specify the number of milliseconds the function must pause before sending the next key stroke. The {wait <milliseconds>} clause must be typed in braces. To send a special printable key such as {, }, ^, %, +, (, or), surround it using braces {}. When sending key combinations that include the Alt key, send lowercase characters. To repeat a key sequence, use the syntax {<key string> <number>}. Put a space between the key and the number. To combine a key code with Shift, Alt, or Ctrl, precede the key code with the following symbols: Shift + Alt % Ctrl ^ You can also group keys with parentheses and precede the group with the key code for a Shift, Alt, or Ctrl key.
<return>	0 or 1. A value of 0 specifies that the function returns immediately. When the function returns immediately, all key information is set up in the buffer to be sent at one time. A value of 1 specifies that the function does not return immediately and waits until all key messages are fully processed before it returns.

EXAMPLES

```
to handle buttonClick
    run "Notepad.exe"
    get sendKeys("Hello World",1)
end
```

sendNotifyAfter

Script Control

SYNTAX

```
sendNotifyAfter <message> [<parameters>] [to <object>]
```

DESCRIPTION

Executes the notifyAfter handler associated with a given message. This command is similar to the send command and operates in the context of the target window.

You must specify an object that contains a notifyAfter handler or the command will not execute.

Executing the sendNotifyAfter command resets the value of target in the handler to the uniqueName of the object that receives the message.

PARAMETERS

PARAMETER	DESCRIPTION
<message>	A built-in or user-defined message.
<parameters>	A list of expressions to be sent with the message.
<object>	The unique name for an object, or system.

EXAMPLES

```
sendNotifyAfter idle to field "timer" of this page
```

sendNotifyBefore

Script Control

SYNTAX `sendNotifyBefore <message> [<parameters>] [to <object>]`

DESCRIPTION Executes the `notifyBefore` handler associated with a given message. This command is similar to the `send` command and operates in the context of the target window.

You must specify an object that contains a `notifyBefore` handler or the command will not execute.

Executing the `sendNotifyBefore` command resets the value of `target` in the handler to the uniqueName of the object that receives the message.

PARAMETERS

PARAMETER	DESCRIPTION
`<message>`	A built-in or user-defined message.
`<parameters>`	A list of expressions to be sent with the message.
`<object>`	The unique name for an object, or system.

EXAMPLES

```
sendNotifyBefore idle to field "timer" of this page
```

sendToolBookMessages

Property

DESCRIPTION Determines whether a control sends ToolBook messages or their OCX/VBX equivalents in response to user events.

NOTES You can get or set this property.

If an OCX/VBX message does not appear in the table below (such as `GotFocus` or `LostFocus`), the message is not translated into an equivalent ToolBook message; you must write handlers for the OCX/VBX message.

TOOLBOOK MESSAGE	OCX/VBX MESSAGE EQUIVALENT
`buttonClick`	`Click`
`buttonDoubleClick`	`DblClick`
`buttonDown`	`MouseDown <button> <isShift>,<xPos>,<yPos>`
`buttonUp`	`MouseUp <button> <isShift>,<xPos>,<yPos>`
`keyChar`	`KeyPress <key>`
`keyDown`	`KeyDown <keyCode>,<isShift>`
`keyUp`	`KeyUp <keyCode>,<isShift>`

The following ToolBook messages are always sent regardless of the `sendToolBookMessages` setting: `moved`, `destroy`, and `sized`.

ToolBook can translate OCX/VBX messages to standard ToolBook messages, only if the control sends the standard form (that is a message with the standard set of parameters) of the OCX/VBX message.

VALUES `True` or `false`; the default is `false`.

If `true`, OCX/VBX controls send ToolBook messages.

If `false`, they send the OCX/VBX messages listed in the table above.

EXAMPLES

```
sendToolBookMessages of self = true
```

```
sendToolBookMessages of self = false
```

set

Assigning Values

SYNTAX `set [the] <container> [of <object>] to <value|container> [of <object>]`

DESCRIPTION Changes the value of a specified property or other container.

NOTES You can also use the = assignment operator to assign a value to a container.

If you attempt to use the `set` command with a property name that is not a standard property, ToolBook will search the object hierarchy for a corresponding `to set` handler for that property name. If no such handler exists, ToolBook then treats the name as a `user property` and sets its value.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	A valid expression.
`<container>`	A reference to a container.

EXAMPLES

```
set the caption of button "start" to "Start Again"

-- These two do the same thing
x = 5
set x to 5

set ASYM_BeenHere of page "printing template" to null
```

setCurrentDirectory()

TBDOS.DLL File Function (Directory)

SYNTAX `setCurrentDirectory(<directory name>)`

DESCRIPTION Makes the specified directory the current working directory on a drive. Each drive maintains its own current directory.

RETURNS

`1` Function was successful.
`-3` Specified path was invalid.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    INT setCurrentDirectory(STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `setCurrentDirectory32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<directory name>`	The path of the directory to be set as the current directory.

EXAMPLES

```
get setCurrentDirectory("c:\tb\training")
```

setCurrentDirectory32()

TBFILE32.DLL File Function (Directory)

SYNTAX `setCurrentDirectory32(<directory name>)`

DESCRIPTION Makes the specified directory the current working directory on a drive. Each drive maintains its own current directory.

RETURNS `1` Function was successful.

Otherwise, the function returns one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    INT setCurrentDirectory32(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<directory name>`	The path of the directory to be set as the current directory.

EXAMPLES

```
get setCurrentDirectory32("c:\tb\training")
```

setCurrentDrive()

TBDOS.DLL File Function (Drive)

SYNTAX `setCurrentDrive(<drive letter>)`

DESCRIPTION Makes the specified disk drive the current drive.

RETURNS

`1` Function was successful.
`-1` Error occurred.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    INT setCurrentDrive(STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `setCurrentDrive32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<drive letter>`	The letter designating the disk drive.

EXAMPLES

```
get setCurrentDrive("a")
```

setCurrentDrive32()

TBFILE32.DLL File Function (Drive)

SYNTAX `setCurrentDrive32(<drive letter>)`

DESCRIPTION Makes the specified disk drive the current drive.

RETURNS `1` Function was successful.

Otherwise, the function returns one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    INT setCurrentDrive32(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<drive letter>`	The letter designating the disk drive.

EXAMPLES

```
get setCurrentDrive32("a")
```

setCustomColors()

TBDLG.DLL Color Functions

SYNTAX `setCustomColors(<index>,<colors>)`

DESCRIPTION Sets the colors for the custom color palette used by the `colorPaletteDlg()` function.

If more than one application is using the `colorPaletteDlg()` function in `TBDLG.DLL`, you can use the `getCustomColors()` and `setCustomColors()` functions to reset the custom color palette for each instance in which it is used.

To reset the custom color palette for a particular instance, call the `getCustomColors()` function just after the Color dialog box closes and store the returned string of values in a variable (or the text of a field). Just before the Color dialog box is shown again, call the `setCustomColors()` function and pass the previously stored values as an argument.

RETURNS If no error occurs, the function returns `1`.

`-1` Index supplied is out of the `1` to `16` range.
`-2` Fewer than three values are supplied for a color.
`-3` Color list string is too long.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    INT setCustomColors(INT,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<index>	An integer (1 to 16) that represents an index to a specific value in the array for the custom color palette. The <index> parameter tells the function where to begin filling the custom color palette. Each color space in the palette has an index number, from 1 to 16.
<colors>	A string that contains a comma-separated list of RGB values. Each string of three values can be CRLF-delimited. Each set of three values represents one color. The <colors> parameter accepts strings of values that are separated by commas and delimited by CRLFs to allow you to use the text of a field rather than passing a literal string. You can also save the values returned by getCustomColors() and pass these values at <index> 1 to set the custom color palette back to its original state.

EXAMPLES

```
get setCustomColors(5,"255,0,0")
```

setFileAttributes()

TBDOS.DLL File Function (Attribute)

SYNTAX

```
setFileAttributes(<file name>,<attributes>)
```

DESCRIPTION

Set the file attributes for the specified file.

RETURNS

```
 1  Function was successful.
-2  Specified file was not found.
-3  Specified path was invalid.
-5  Access to the file was denied.
```

NOTES

You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    INT setFileAttributes(STRING,STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function setFileAttributes32() instead.

PARAMETERS

PARAMETER	DESCRIPTION
<file name>	The name of the file with the attributes you want.
<attributes>	A string containing one letter for each attribute to be set: R = read-only S = system H = hidden A = archive

EXAMPLES

```
get setFileAttributes(fName,"r")
```

setFileAttributes32()

TBFILE32.DLL File Function (Attribute)

SYNTAX

```
setFileAttributes32(<file name>,<attributes>)
```

DESCRIPTION

Set the file attributes for the specified file.

RETURNS

```
 1  Function was successful.
-2  Specified file was not found.
-3  Specified path was invalid.
-5  Access to the file was denied.
```

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    INT setFileAttributes32(STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<file name>	The name of the file with the attributes you want.
<attributes>	A string containing one letter for each attribute to be set: R = read-only S = system H = hidden A = archive

EXAMPLES

```
get setFileAttributes32(fName,"r")
```

setFileDate()

TBDOS.DLL File Function (Attribute)

SYNTAX

```
setFileDate(<path>,<time>,<date>)
```

DESCRIPTION Sets the date and time stamp for a specified file.

RETURNS If no error occurs, `setFileDate()` returns `1`.

Otherwise, the function one of these error values:

- `-2` File was found in path, but not in current directory.
- `-3` Specified file was not found, or the path and directory were not valid.
- `-4` Invalid parameter.
- `-18` Illegal wildcard in file specification.
- `-20` Memory allocation error.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    INT setFileDate(STRING,STRING,STRING)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `setFileDate32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
<path>	The path and file name for which you want to set the time and date.
<time>	The new time in the following ToolBook format: "h24:min:sec".
<date>	The new date in the following ToolBook format: "m/d/y".

EXAMPLES

```
to handle buttonClick
    fTime = sysTime
    format time fTime as "h24:min:sec"
    fDate = sysDate
    format date fDate as "m/d/y"
    get setFileDate("c:\file.txt",fTime,fDate)
end
```

setFileDate32()

TBFILE32.DLL File Function (Attribute)

SYNTAX `setFileDate32(<path>,<time>,<date>)`

DESCRIPTION Sets the date and time stamp for a specified file.

RETURNS If no error occurs, `setFileDate32()` returns `1`. Otherwise, the function returns `null` and the value of `sysError` is set to one of the values listed in the `TBFILE32 Error Code Table`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    INT setFileDate32(STRING,STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<path>`	The path and file name for which you want to set the time and date.
`<time>`	The new time in the following ToolBook format: "`h24:min:sec`".
`<date>`	The new date in the following ToolBook format: "`m/d/y`".

EXAMPLES

```
to handle buttonClick
    fTime = sysTime
    format time fTime as "h24:min:sec"
    fDate = sysDate
    format date fDate as "m/d/y"
    get setFileDate32("c:\file.txt",fTime,fDate)
end
```

setIniVar()

TBWIN.DLL INI Functions

SYNTAX `setIniVar(<section name>,<item name>,<new value>,<file name>)`

DESCRIPTION Sets the value of the specified item in the specified section of any `.INI` file to the specified new value.

RETURNS If no error occurs this function returns `1`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    WORD setIniVar(STRING,STRING,STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<section name>`	The name of the section in the `.INI` file that contains the item whose value you want to set. This parameter is any expression that evaluates to a valid section name in the `.INI` file. If that section does not exist, it is added to the specified `.INI` file.
`<item name>`	The name of the item with the value you want to set.
`<new value>`	The new value you want to set for the specified item.
`<file name>`	The name of the `.INI` file.

EXAMPLES

```
get setIniVar("General","lastScore",curScore,"student005.ini")
```

setMenuHelpText()

Menu Command/Function

SYNTAX

```
setMenuHelpText(<name | alias> in [<menu reference] in <menu
  reference]...],<new help text>[,<viewer reference>])
```

DESCRIPTION

Changes the `help text` assigned to a menu.

The menu must use a unique `name` or `alias`.

The help text appears in the status bar when a user moves the cursor onto the menu.

NOTES

If the menu does not exist, an `Execution Suspend` error will be displayed.

PARAMETERS

PARAMETER	DESCRIPTION
`<name | alias>`	The unique `name` or `alias` of a menu on the menu bar of the target window or viewer specified as the `<viewer reference>`. Use the in `<menu reference>` parameter to refer to a submenu.
`<new help text>`	A string of text to be assigned as the new help text of the specified menu.
`<viewer reference>`	A valid reference to the viewer that owns the menu bar resource with the menu. This parameter is optional.

setMenuItemHelpText()

Menu Command/Function

SYNTAX

```
setMenuItemHelpText(<name | alias> in [<menu reference] in <menu
  reference]...],<new help text>[,<viewer reference])
```

DESCRIPTION

Changes the `help text` assigned to a menu item.

The menu item must use a unique `name` or `alias`.

The help text appears in the status bar when a user moves the cursor onto that menu item.

NOTES

If the menu item does not exist, an `Execution Suspend` error will be displayed.

PARAMETERS

PARAMETER	DESCRIPTION
`<name | alias>`	The unique `name` or `alias` of a menu item on the menu bar of the target window or viewer specified as the `<viewer reference>`. Use the in `<menu reference>` parameter to refer to a submenu.
`<new help text>`	A string of text to be assigned as the new help text of the specified menu item.
`<viewer reference>`	A valid reference to the viewer that owns the menu bar resource with the menu. This parameter is optional.

EXAMPLES

```
-- Changes the help text of a menu item with the alias playMedia to
-- an appropriate value depending on the state of a system variable
to handle enterMenu mnName, mnAlias
    system logical s_isSoundPlaying
    if mnName is "Media Controls"
        if s_isSoundPlaying
            get setMenuItemHelpText("playMedia","Stops music")
        else
            get setMenuItemHelpText("playMedia","Starts music")
        end
    end
end
```

setMenuItemName()

Menu Command/Function

SYNTAX

```
setMenuItemName(<name | alias> in [<menu reference] in <menu
  reference]...],<new name>[,<viewer reference>])
```

DESCRIPTION

Changes the name of a menu item.

The `name` is the `text` of the menu item that is displayed on the menu bar.

NOTES

If the menu item does not exist, an `Execution Suspend` error will be displayed.

PARAMETERS

PARAMETER	DESCRIPTION
`<name | alias>`	The unique `name` or `alias` of a menu item on the menu bar of the target window or viewer specified as the `<viewer reference>`. Use the in `<menu reference>` parameter to refer to a submenu.
`<new name>`	The new name to be assigned to the specified menu item.
`<viewer reference>`	A valid reference to the viewer that owns the menu bar resource with the menu. This parameter is optional.

EXAMPLES

```
-- Changes the name of a menu item with the alias playMedia to
-- an appropriate value depending on the state of a system variable
to handle enterMenu mnName, mnAlias
    system logical s_isSoundPlaying
    if mnName is "Media Controls"
        if s_isSoundPlaying
            get setMenuItemName("playMedia","Stop playing")
        else
            get setMenuItemName("playMedia","Start playing")
        end
    end
end
```

setMenuName()

Menu Command/Function

SYNTAX

```
setMenuName(<name | alias> in [<menu reference] in <menu
  reference]...],<new name>[,<viewer reference>])
```

DESCRIPTION

Changes the name of a menu.

The `name` is the `text` of the menu that is displayed on the menu bar.

NOTES

If the menu does not exist, an `Execution Suspend` error will be displayed.

PARAMETERS

PARAMETER	DESCRIPTION
`<name | alias>`	The unique `name` or `alias` of a menu on the menu bar of the target window or viewer specified as the `<viewer reference>`. Use the in `<menu reference>` parameter to refer to a submenu.
`<new name>`	The new name to be assigned to the specified menu.
`<viewer reference>`	A valid reference to the viewer that owns the menu bar resource with the menu. This parameter is optional.

EXAMPLES

```
get setMenuName("Help","Assistance",mainWindow)
```

setProperty()

Object Function

SYNTAX `setProperty(<objRef>,<propertyName>,<newValue>)`

DESCRIPTION Sets the value of an object's property to a new value.

RETURNS Returns the property's previous value, or `NULL` if the property does not exist.

NOTES When using `setProperty()` or `getProperty()`, the object's property will be accessed directly.

You cannot use the `to get` or `to set` handler structure to intercept these functions in OpenScript.

PARAMETERS

PARAMETER	DESCRIPTION
`<objRef>`	A reference to the resource.
`<propertyName>`	A string representing the name of the property to set.
`<newValue>`	A string representing the value of the property.

EXAMPLES

```
-- This handler clears all property values with
-- names that begin with _ASYM
to handle clearAsymProps pObj
    propList = propertyList(pObj)
    while propList <> NULL
        pop propList into propName
        if chars 1 to 5 of propName = "_ASYM"
            get setProperty(pObj,propName,NULL)
        end
    end
end
```

setSystemDate()

TBDOS.DLL Date Function

SYNTAX `setSystemDate(<month>,<day>,<year>)`

DESCRIPTION Sets the system clock to the specified date.

RETURNS If no error occurs, `setSystemDate()` returns `1`.

Otherwise, the function returns `0`.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    INT setSystemDate(INT,INT,INT)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `setSystemDate32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<month>`	An integer from `1` through `12` that represents the new value for the month.
`<day>`	An integer from `1` through `31` that represents the new value for the day.
`<year>`	A 4 digit integer from `1980` through `2099` that represents the new value for the year.

EXAMPLES

```
get setSystemDate(7,1,2002)
```

setSystemDate32()

TBFILE32.DLL Date Function

SYNTAX `setSystemDate32(<month>,<day>,<year>)`

DESCRIPTION Sets the system clock to the specified date.

RETURNS If no error occurs, `setSystemDate()` returns `1`.

Otherwise, the function returns `0`.

NOTES As of ToolBook 8.5 `getFileAttributes32()` is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    INT setSystemDate32(INT,INT,INT)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<month>`	An integer from `1` through `12` that represents the new value for the month.
`<day>`	An integer from `1` through `31` that represents the new value for the day.
`<year>`	A 4 digit integer from `1980` through `2099` that represents the new value for the year.

EXAMPLES `get setSystemDate32(7,1,2002)`

setSystemTime()

TBDOS.DLL Time Function

SYNTAX `setSystemTime(<hour>,<minute>,<seconds>)`

DESCRIPTION Sets the system clock to the specified time.

RETURNS If no error occurs, `setSystemTime()` returns `1`.

Otherwise, the function returns `0`.

NOTES You will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdos.dll"
    INT setSystemTime(INT,INT,INT)
end
```

This function will not exist in version 9 and higher of ToolBook, so it is recommended that you use the function `setSystemTime32()` instead.

PARAMETERS

PARAMETER	DESCRIPTION
`<hour>`	An integer from `0` through `23` that represents the new value for the hour, based on a `24`-hour clock.
`<minute>`	An integer from `0` through `59` that represents the new value for the minute.
`<seconds>`	An integer from `0` through `59` that represents the new value for the seconds.

EXAMPLES `get setSystemTime(18,30,15)`

setSystemTime32()

TBFILE32.DLL Time Function

SYNTAX `setSystemTime32(<hour>,<minute>,<seconds>)`

DESCRIPTION Sets the system clock to the specified time.

RETURNS If no error occurs, `setSystemTime()` returns `1`.

Otherwise, the function returns `0`.

NOTES As of ToolBook 8.5 `getFileAttributes32()` is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbfile32.dll"
    INT setSystemTime32(INT,INT,INT)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<hour>`	An integer from `0` through `23` that represents the new value for the hour, based on a `24`-hour clock.
`<minute>`	An integer from `0` through `59` that represents the new value for the minute.
`<seconds>`	An integer from `0` through `59` that represents the new value for the seconds.

EXAMPLES

```
get setSystemTime32(18,30,15)
```

setWinIniVar()

TBWIN.DLL INI Functions

SYNTAX `setWinIniVar(<section name>,<item name>,<new value>)`

DESCRIPTION Sets the value of the specified item in the specified section of the `WIN.INI` file to the specified new value.

RETURNS If no error occurs this function returns `1`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    WORD setWinIniVar(STRING,STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<section name>`	The name of the section in the `WIN.INI` file that contains the item whose value you want to set. This parameter is any expression that evaluates to a valid `WIN.INI` section name. If that section does not exist, it is added to the `WIN.INI` file.
`<item name>`	The name of the item with the value you want to set.
`<new value>`	The new value you want to set for the specified item.

EXAMPLES

```
get setWinIniVar("General","WallPaper","blue.bmp")
```

seventh

Ordinal Numbers

DESCRIPTION A value used to indicate the seventh item in a sequence or list. Also used to indicate relative position of pages or backgrounds.

EXAMPLES

```
-- The following paired examples show how you can use ordinal
-- references to make OpenScript easier to read. As you can see it is
-- possible to write your scripts with or without ordinal references.
go to the seventh page of this book
go to page 7 of this book

if seventh character of str = "X"
if character 7 of str = "X"

seventh word of text of field "lesson" = "Hello"
word 7 of text of field "lesson" = "Hello"

put "Cancelled:" into seventh character of name of button id 16
put "Cancelled:" into character 7 of name of button id 16
```

sharedScript

Resource Type

DESCRIPTION The resource type name for a `script` resource.

Script resources are referenced by `name` or `ID` with the `sharedScript` keyword. The `ID` is assigned by the ToolBook system; however, you can assign any `name` to the resource.

You can set the `sharedScript` property of an object to a `sharedScript` script resource. You can also set the `script` property of a `sharedScript` resource.

EXAMPLES

```
get resourceInfo of sharedScript "tabs"

script of sharedScript "departure" = script of button "exit"
```

sharedScript

Property

DESCRIPTION A property of all ToolBook object types that specifies a reference to its assigned `shared script`.

NOTES You can get or set this property.

VALUES The value of the property `sharedScript` is a reference to a single `shared script` resource assigned to the object.

If `null` the object has no `sharedScript` currently assigned to it.

EXAMPLES

```
if sharedScript of button "Exit" = null
    sharedScript of button "Exit" = sharedScript "standard exit code"
end
```

show

Viewer & Object Command

SYNTAX

```
show <object> [at <location>]
show <viewer reference> [as <mode>] [at <position>]
```

DESCRIPTION Shows the specified object if it was previously `hidden`.

Using this command is similar to setting the object's `visible` property to `true`.

If `<location>` is specified, the object is moved to the location before it is shown.

NOTES For `viewers`, the show command automatically opens the viewer if it is not already open, then shows it. You can assign specific behavior that defines how the viewer is shown by using the `<mode>` and `<position>` parameters. The as `<mode>` parameter specifies if the viewer is shown in `modal` state, or if the viewer is shown but is not active.

When a viewer is shown with the `as modal` parameter, all other viewers are `disabled` until the viewer is `hidden` or `closed`. Modal windows (often used as dialog boxes or message boxes) stop all other scripts from running until dismissed by the user. ToolBook's `Request` and `Ask` dialog boxes are examples of modal windows.

When a viewer is shown with the as `notActive` parameter, the viewer is shown, but is not active. Modal windows must be active when they are shown.

PARAMETERS

PARAMETER	DESCRIPTION
`<object>`	The unique name of an object, or the object type name for a built-in ToolBook window or palette.
`<location>`	A list of two numbers: in `page units` for all ToolBook objects except viewers; in `pixels` for viewers and ToolBook system windows and palettes.
`<viewer reference>`	A valid reference to a viewer or a list of viewers.
`<mode>`	`Modal` or `notActive`. If `modal`, the viewer remains active in the current instance of ToolBook until the user closes it. If `notActive`, the viewer is not active when it is shown.
`<position>`	A list of two numbers in `pixels` that define the upper-left corner of the viewer to be shown.

EXAMPLES

```
to handle wrongAnswer whichPage
    conditions
        when whichPage is 2
            show field "Hint1"
        when whichPage is 3
            show field "Hint2"
    end
end

-- Shows a viewer as a modal dialog box
show viewer "Message1" as modal

-- Shows a viewer in upper-left corner of screen
show viewer "Message1" at 0,0
```

showHotwords

Hotword Message

DESCRIPTION Sent when `Show Hotwords` is chosen from the `View` menu.

You can also `send` this message using the `send showHotwords` statement. ToolBook's default response is to toggle the value of the `sysHotwordsShown` property.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle buttonClick
    send showHotwords
end

to handle showHotwords
    request "Sorry, you are not permitted to perform this action."
end
```

shown

Notification Message

DESCRIPTION Sent to a `viewer` when it is shown.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle shown
    position of self = 100,100
    forward
end
```

shownBy

Property

DESCRIPTION A page property that specifies a list of any viewers currently displaying the page.

NOTES This is a read only property and cannot be set.

VALUES A comma separated list of one or more unique names of each viewer displaying the page.

If no viewer is currently displaying the page the value will be `null`.

EXAMPLES

```
-- Hides any viewers showing page "contents"
hide shownBy of page "contents"
```

sin()

Trigonometric Values

SYNTAX

```
sin(<angle>)
```

DESCRIPTION Returns the sine of an angle that is measured in radians.

The formula for converting degrees to radians is `radians = degrees * (pi/180)`.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<angle>	An expression that yields a number.

EXAMPLES

```
x = sin(pi)         -- equals 1.22E-16, which is approximately 0
                    -- (zero). The sine of pi is zero.
x = sin(pi/2)       -- equals 1
x = sin(30*pi/180)  -- equals 0.5, the sine of 30 degrees
```

singleLine

Property

DESCRIPTION A property of a field or recordfield specifying that only one line of text appears in the field or recordfield.

NOTES You can get or set this property.

This has the same effect as setting the `fieldType` to `singleLineWrap`.

VALUES `True` or `false`.

The default is `false`.

EXAMPLES

```
singleLine of field "help" = true
```

sinh()

Trigonometric Values

SYNTAX `sinh(<angle>)`

DESCRIPTION Returns the hyperbolic sine of an angle that is measured in radians.

The formula for converting degrees to radians is `radians = degrees * (pi/180)`.

PARAMETERS

PARAMETER	DESCRIPTION
`<angle>`	An expression that yields a number.

EXAMPLES

```
x = sinh(1)   -- equals 1.175201194
x = sinh(-1)  -- equals -1.175201194
```

sixth

Ordinal Numbers

DESCRIPTION A value used to indicate the sixth item in a sequence or list. Also used to indicate relative position of pages or backgrounds.

EXAMPLES

```
-- The following paired examples show how you can use ordinal
-- references to make OpenScript easier to read. As you can see it is
-- possible to write your scripts with or without ordinal references.
go to the sixth page of this book
go to page 6 of this book

if sixth character of str = "X"
if character 6 of str = "X"

sixth word of text of field "lesson" = "Hello"
word 6 of text of field "lesson" = "Hello"

put "Cancelled:" into sixth character of name of button id 16
put "Cancelled:" into character 6 of name of button id 16
```

size

Property

DESCRIPTION A property of almost all ToolBook object types that specifies the current size of that object. The size of an object refers to a combination of width and height.

NOTES You can get or set this property.

A `page` does not have a `size` property since its `size` is determined by the background it lives on, so the background `size` is what you need to change in order to change the apparent `size` of a page. A `hotword` does not have a `size` property.

VALUES A list of two positive whole numbers that specify `width` and `height`. The value of the `size` property for all objects, except `viewers`, is in `page units`. The value of the `size` property for `viewers` is in `pixels`.

For `books` the default `size` for a new book is the value of `startupWidth` and the value of `startupHeight`. Setting this value is the same as specifying a `size` in the Page Size dialog box.

For `backgrounds`, if a background's `size` property is set to `0,0` the book's size property is used for the pages of that background. The size property for new `backgrounds` is `0,0` by default. For `viewers` the initial value for `size` is determined by the setting for `defaultClientSize`.

EXAMPLES

```
-- resize ellipse "target" to be exactly 100,200
size of ellipse "target" = 100,200

-- ensure size of rectangle "a" is the same as rectangle "b"
size of rectangle "a" = size of rectangle "b"
```

sized

Notification Message

DESCRIPTION Sent at `Author` level to an object when its `bounds`, `size`, or `vertices` property changes.

If a `group` is resized, only the group receives the `sized` message, not its individual members.

For `viewers`, the `sized` message is sent whenever a viewer is resized at either `Author` or `Reader` level. The `sized` message is also sent to a viewer when the viewer is `maximized` or `minimized`, and when it is restored to `normal` after being `maximized` or `minimized`.

If you choose the `Undo` command from the `Edit` menu after resizing an object, the `sized` message is **not** sent to the object when ToolBook restores the object to its original size.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
-- Prevents an individual object from being resized
to handle sized
    if target is rectangle "box" and bounds of target <> origBounds of target
        bounds of target = origBounds of target
    else
        forward
    end
end

-- Handles the sized message for viewers at Reader level
to handle sized
    if sysLevel is Reader
        send rearrangeChildWindows
    end
    forward
end
```

sizeToPage

Window Sizing

DESCRIPTION Sent to the page when you choose `Size To Page` from the `View` menu. You can also send this message using the `send sizeToPage` statement.

The `sizeToPage` message is automatically sent to the page each time a book is opened.

ToolBook's default response is to expand or contract the ToolBook Main window to the page size of the current book or to the maximum window size, whichever is smaller. The sizing occurs *before* any enter event messages are sent.

If the focus is in a viewer that does not have a menu bar, you can press the `F11` key to send the `sizeToPage` message to the page displayed in the viewer.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
to handle enterPage
    send sizeToPage
    forward
end
```

skipNavigation

Property

DESCRIPTION A page property that specifies whether the page is included in normal page navigation at `Reader` level.

NOTES You can get or set this property.

When the `skipNavigation` property is `true`, navigation that occurs as the result of implicit statements such as `send next`, `send previous`, `send first`, `send last`, or `flip` does not include the page.

Navigation can only occur with explicit statements that go to another page, such as `go to page "topics"`.

For example, if ToolBook reaches a page with `skipNavigation` set to `true` as the result of the `send next` statement, ToolBook skips that page and goes to the next page.

If you create a dialog box by displaying a page with buttons and other controls in a viewer, you can exclude it from normal page navigation by setting the page's `skipNavigation` property to `true`.

To find pages in a book for which the `skipNavigation` property is not `true`, you can use the functions `firstPage()`, `lastPage()`, `nextPage()`, and `previousPage()`.

VALUES `True` or `false`; the default is `false`.

If `false`, the page is included in normal page navigation.

If `true`, the page is not included in navigation resulting from the `next`, `previous`, `last`, or `first` messages being sent or the `flip` command being executed.

solidColorsEnabled

Property

DESCRIPTION A property of a book or picture object that specifies whether objects in the book are displayed using solid object colors or dithered object colors.

NOTES When you set this property to `true` for a book or an individual picture object, ToolBook attempts to provide the best match for picture colors to the current palette, rather than `dithering` the colors.

NOTE: The use of `solidColorsEnabled` is pretty much a holdover from the days when computers commonly were configured to only display 256 [`8-bit`] colors. If configured to display `16-bit`, `24-bit`, or higher color, then the `solidColorsEnabled` has no effect.

VALUES `True` or `false`. The default is `true`.

If `true`, ToolBook uses solid object colors. The solid object colors appear in the Color Tray in `32` standard colors and `64` custom solid colors. Each of the `64` custom colors can be edited.

If `false`, ToolBook uses dithered object colors.

EXAMPLES

```
solidColorsEnabled of this book = false
```

sort

Data Manipulation

SYNTAX

```
sort [pages <number> to <number>] [by <sort key> [,<sort key> ... ]]
```

DESCRIPTION Sorts specified pages that share the current background, based on the specified sort keys. You can specify as many sort keys as you want.

NOTES The sort takes place in the `target window`. Unlike the `Sort Pages` menu item, which sorts pages only by the content of their recordfields, the `sort` command can sort pages by any expression that can be evaluated in the context of a page.

For example, use the `sort` command to sort pages based on page name.

Characters are sorted according to their `ANSI` values, but are not case sensitive.

PARAMETERS

PARAMETER	DESCRIPTION
`<number>`	A positive integer that specifies the pages to be sorted. The default is all pages that share the current background.
`<sort key>`	A statement in this form: `[<order>] <type> <sort expression>`
`<order>`	`Ascending` or `descending`; the default is `ascending`.
`<type>`	`Date`, `name`, `text`, or `number`, to specify the type of sort.
`<sort expression>`	Any expression that can be evaluated in the context of a page, such as the page name. The default is the text of the recordfield with the lowest layer number on the current background. To see how ToolBook evaluates a sort expression, go to any page to be included in the sort and execute the `sort` expression in the `Command window`.

EXAMPLES

```
--Sorts pages 1 to 10 numerically by contents of a recordfield
sort pages 1 to 10 by ascending number text of recordfield "zipcode"

--Sorts based on the content of two recordfields
sort pages 10 to 20 by ascending text text of recordfield "Name", \
    descending number last word of text of recordfield "CityState"
```

sortItems

Property

DESCRIPTION A combobox property that specifies whether the items in its dropdown list are sorted alphabetically at `Reader` level.

NOTES You can get or set this property.

VALUES `True` or `false`.

The default is `false`.

sortList()

TBDLG.DLL String Functions

SYNTAX

```
sortList(<list>)
```

DESCRIPTION Alphabetically sorts the list items of a string.

This function sorts a maximum of `5000` list items.

RETURNS Returns the sorted list items.

If an error occurs then `null` is returned and `sysError` is set to a negative value.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING sortList(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<list>	An expression that results in a string value.

EXAMPLES

```
to handle buttonClick
    -- Sort list of flower names and remove duplicates
    x = sortList(text of field "namesOfFlowers")
    step k from itemCount(x) to 2 by –1
        if item k of x = item k-1 of x
            clear item k of x
        end
    end
    text of field "namesOfFlowers" = x
end
```

sortTextlines()

TBDLG.DLL String Functions

SYNTAX

```
sortTextlines(<textlines>)
```

DESCRIPTION

Sorts the textlines of a string by ASCII value (alphabetically). This function sorts a maximum of 5000 textlines.

RETURNS

Returns the sorted textlines. If an error occurs then null is returned and sysError is set to a negative value.

NOTES

As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING sortTextlines(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<line>	An expression that results in a string value.
<text>	An expression that results in a string value.

EXAMPLES

```
to handle buttonClick
    -- Sort a textline list of flower names and remove duplicates
    x = sortTextlines(text of field "namesOfFlowers")
    step k from textlineCount(x) to 2 by –1
        if textline k of x = textline k-1 of x
            clear textline k of x
        end
    end
    text of field "namesOfFlowers" = x
end
```

space

Spacing Constants

DESCRIPTION

Represents the space ANSI character. This is equivalent to ANSI 32 as well as " ".

EXAMPLES

```
-- remove leading spaces from a text string
ask "What is your Full Name"
fullName = it
while first character of fullName = space
    clear first char of fullName
end

-- both of these are valid
a = b & space & c
a = b & " " & c
```

spacing

Property

DESCRIPTION A field or recordfield property that specifies the line spacing for text.

NOTES You can get or set this property.

Notice that you can only set one spacing per field, rather than one per paragraph (textline) as you can in Microsoft Word. This is an inherent limitation of ToolBook fields.

VALUES `1`, `1.5`, or `2`. The default is the value of `sysLineSpacing`.

sqrt()

Arithmetic Values

SYNTAX `sqrt(<number>)`

DESCRIPTION Returns the `square root` of a number.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<number>`	An expression that results in a number.

EXAMPLES

```
x = 12        -- start with 12
y = x ^ 2     -- calculate the Square      - results in 144
val = sqrt(y) -- calculate the Square Root - results in 12
```

stage

Object Type

DESCRIPTION The object type name for a multimedia `stage`.

NOTES A `stage` provides a location and frame for playing visual media. You can customize the appearance of its frame, and create transition effects for the media it displays.

You can play any visual media type (video overlay, digital video, animation, bitmaps, videodisc, or Photo CD) in a stage.

By playing visual media in a stage, you eliminate the need to write scripts that position and size windows.

The stage can be automatically sized, which eliminates device-dependency problems across different systems.

stageAnchor

Property

DESCRIPTION A stage property that specifies the anchor position of a stage.

NOTES You can get or set this property.

Useful when working with a stage that stretches to fit the media it displays (if its `stageSizing` property is set to `stretchStage`), or how media is positioned inside a stage (if its `stageSizing` property is set to `clipMedia`).

Because the stage may resize differently for different display devices, you can specify that the stage is anchored at a particular corner or at its center.

VALUES Possible values: `bottomLeft`, `bottomRight`, `center`, `topLeft`, or `topRight`.

The default is `center`.

EXAMPLES

```
stageAnchor of stage "player" = "topLeft"
```

stageSizing

Property

DESCRIPTION A stage property that specifies how ToolBook resolves differences in size between the stage's display area and media to be played.

NOTES You can get or set this property.

VALUES `CenterMedia`, `clipMedia`, `stretchMedia`, or `stretchStage`.

The default is `centerMedia`.

If `centerMedia`, media are centered in stage's display area. If a clip is larger than the display area, its edges are cropped evenly.

If `clipMedia`, media are positioned according to the value of the `stageAnchor` property. If the clip is still too large, it is further cropped on the edges opposite the anchor.

If `stretchMedia`, media are sized to fit the stage's display area.

If `stretchStage`, the stage is positioned and sized according to the values of its `stageAnchor` and `mediaSize` properties.

EXAMPLES

```
stageSizing of stage "player" = "stretchMedia"
```

start spooler

Control Structure

SYNTAX

```
start spooler
    <statements>
end [spooler]
```

DESCRIPTION Starts a print job that prints the pages specified by the statements executed between the `start spooler` and `end [spooler]` statements.

NOTES The `print` command can only be used within a `start spooler` structure.

You cannot nest `start spooler` control structures.

Use the `go` command within the `start spooler` control structure to print nonconsecutive page ranges.

The `start spooler` control ONLY works for pages in the main window. This means that printing the pages contained in a viewer other than the main window will be difficult. To do so, you will need to navigate the main window to the same page as shown in the viewer, and then issue a print command, after which you will need to navigate the main window back to where it previously was.

PARAMETERS

PARAMETER	DESCRIPTION
`<statements>`	One or more OpenScript statements including a `print` command.

EXAMPLES

```
--Prints all pages if the word "Complete" is in book caption
start spooler
    if the caption of this book contains "Complete"
        print all pages
    end
end

--Prints only odd pages in a book
start spooler
    step i from 1 to pageCount of this book by 2
        go to page i
        print
    end
end
```

startup3Dinterface

System Property

DESCRIPTION Specifies the default value of `sys3DInterface` for the ToolBook instance, as defined in the `startup3DInterface=` line of the `INSTRUCTOR.INI` file.

NOTES Setting this property modifies the entry in the `INSTRUCTOR.INI` file for ToolBook.

VALUES `True` or `false`.

The default is `true`.

startupBook

System Property

DESCRIPTION Specifies the name of the book that opens automatically when the user runs ToolBook, as defined in the `startupBook=` line of the `INSTRUCTOR.INI` file.

VALUES Any legal ToolBook file name, including the path if necessary.

The default is `null`.

startupDrawDirect

System Property

DESCRIPTION Specifies the default value of `sysDrawDirect` for the ToolBook instance, as defined in the `startupDrawDirect=` line of the `INSTRUCTOR.INI` file.

NOTES Setting this property modifies the entry in the `INSTRUCTOR.INI` file for ToolBook.

VALUES `True` or `false`.

The default is `true`.

startupGifInterlaced

System Property

DESCRIPTION Specifies the default value of sysGifInterlaced which controls whether a bitmap resource will have interlacing turned on or off when it is exported as a `GIF` file.

NOTES Setting this property modifies the entry in the `INSTRUCTOR.INI` file for ToolBook.

VALUES `True` or `false`.

The default is `false`.

startupHeight

System Property

DESCRIPTION Specifies a new book's default page height as defined in the `startupHeight=` line of the `INSTRUCTOR.INI` file.

NOTES Setting this property modifies the entry in the `INSTRUCTOR.INI` file for ToolBook.

VALUES A number from `1` to `12960` in `page units`.

The default is `6000`.

startupIdleDelay

System Property

DESCRIPTION Specifies a initial setting of `sysIdleDelay` when ToolBook is launched, as defined in the `sysIdleDelay=` line of the `INSTRUCTOR.INI` file.

NOTES Setting this property modifies the entry in the `INSTRUCTOR.INI` file for ToolBook.

This property is not supported at runtime.

VALUES A value representing the desired idle delay in milliseconds.

Setting this property to `-1` will cause the behavior of previous versions of ToolBook by effectively causing ToolBook to generate idle messages as fast as possible.

Setting this property to `0` will cause ToolBook to yield to other processes before generating an idle message.

startupReaderRightClick

System Property

DESCRIPTION Specifies the default setting for the `sysReaderRightClick` property for the ToolBook instance, as defined in the `startupReaderRightClick=` line of the `INSTRUCTOR.INI` file.

NOTES Setting this property modifies the entry in the `INSTRUCTOR.INI` file for ToolBook.

VALUES `True` or `false`.

The default is `false`.

startupSysBooks

System Property

DESCRIPTION Specifies the default system books as defined in the `startupSysBooks= line` of the `INSTRUCTOR.INI` file.

NOTES Setting this property modifies the entry in the `INSTRUCTOR.INI` file for ToolBook.

Whenever an instance of ToolBook is opened, the list of files in this .INI setting becomes the value of `sysBooks`.

VALUES Commas separated list of book file names.

The default is `null`.

startupToolTips

System Property

DESCRIPTION Specifies a initial setting of `sysToolTips` when ToolBook is launched, as defined in the `startupToolTips=` line of the `INSTRUCTOR.INI` file.

NOTES Setting this property modifies the entry in the `INSTRUCTOR.INI` file for ToolBook.

VALUES If `startupToolTips` is true, the initial setting of `sysToolTips` is `true`.

If `startupToolTips` is false, the initial setting of `sysToolTips` if `false`.

startupWidth

System Property

DESCRIPTION Specifies a new book's default page width as defined in the `startupWidth=` line of the `INSTRUCTOR.INI` file.

NOTES Setting this property modifies the entry in the `INSTRUCTOR.INI` file for ToolBook.

VALUES A number from `1` to `12960` in `page units`.

The default is `9000`.

state

Property

DESCRIPTION A property of a viewer that specifies the `state` of the viewer.

The `state` refers to whether the window appears minimized as an icon; maximized to fill the entire desktop (if it is a popup window) or its client area (if it is a child window); or in its default size and position.

NOTES You can get or set this property.

Setting a viewer's state property to normal does not activate the viewer. Use the `activate` command to make a viewer the active window.

VALUES `Normal`, `minimized`, `maximized`, or `lockMinimized`.

The default is the setting for `defaultState`.

A lock-minimized window remains in its minimized state; it cannot be changed by the user. You may want to set a viewer's state to `lockMinimized` for an application that runs in the background while other applications are active.

If a viewer's `borderStyle` is set to `dialogFrame`, the viewer's state is always `normal` (ToolBook ignores the settings for `defaultState` and `state`), and only the `Move` and `Close` commands are available from the viewer's Control menu.

EXAMPLES

```
state of mainWindow = "maximized"
state of viewer "help" = "normal"
```

stateChanged

Notification Message

SYNTAX `stateChanged <new state>,<old state>`

DESCRIPTION Sent to a viewer when its `state` changes.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
`<new state>`	The new value of the `state` property: `normal`, `maximized`, `minimized`, or `lockMinimized`.
`<old state>`	The previous value of the `state` property: `normal`, `maximized`, `minimized`, or `lockMinimized`.

EXAMPLES

```
to handle stateChanged newState
    if newState = "minimzed"
        request "To restore the window, double-click its icon."
    end
    forward
end
```

statusBar

System Object

DESCRIPTION The object type name for the status bar located at the bottom of a viewer (including the Main window). The status bar consists of a `caption area`, `status boxes`, `indicators`, and `navigation controls`. The status bar displays system or custom messages in its caption area as long as the status bar is visible.

NOTES For all viewers except the Main window, the status bar is hidden by default. In the Main window, the status bar is shown at `Author` level and hidden at `Reader` level by default.

If the status bar is hidden, all of its elements are hidden as well. Show or hide the status bar in the target window by choosing `Status Bar` from the `View` menu, or by using a `show statusBar` or `hide statusBar` statement or setting its `visible` property.

You can show or hide each element in the status bar using the show or hide command with `statusBox`, `statusControls`, or `statusIndicators`.

To control whether the status bar can be shown in a viewer at either `Author` or `Reader` level, you can set the viewer's `authorStatusBar` or `ReaderStatusBar` property to `true` or `false`.

The status bar automatically resizes itself to the width of the target window. The width of the caption area expands and contracts as necessary to accommodate different window sizes. The widths of the other status bar elements are fixed.

To display custom status information in the status bar, set the `caption` property of the status bar to a string of text.

PROPERTY	VALUES
`caption`	String of up to `255` characters; default is `null`.
`visible`	`True` or `false`. In `Author`-level Main window, default is `true`; at `Reader` level, default is `false`. In other viewers, default is `false`.

COMPONENT	REFERENCE
Status bar	`statusBar`
Caption area	`caption of statusBar`
Mouse position, page selected, and recording indicators	`statusIndicators`
Navigation buttons	`statusControls`
Status box	`statusBox`

statusBox

System Object

DESCRIPTION The object type name for the area on the status bar that displays the current page number and the total number of pages in the book.

NOTES When a user clicks the status box, ToolBook displays a Go To Page box on the status bar that allows the user to navigate to a specific page in the book. You can show or hide the status box by using the `show` or `hide` command or setting the `visible` property of `statusBox`.

statusControls

System Object

DESCRIPTION The object type name for the area on the status bar that displays navigation buttons the user can press to go to the previous or next page in the book.

NOTES You can show or hide the status controls by using the `show` or `hide` command or setting the `visible` property of `statusControls`.

statusIndicators

System Object

DESCRIPTION The object type name for the area on the status bar that displays the mouse position in `page units`, and the page selected indicator.

NOTES The page selected indicator is only visible at `Author` level.

You can show or hide the status indicators by using the `show` or `hide` command or setting the `visible` property of `statusIndicators`.

step

Control Structure

SYNTAX

```
step <variable> from <start> to <finish> [by <steps>]
    <statements>
end [step]
```

DESCRIPTION Executes a block of statements a specified number of times.

The `step` control structure in ToolBook is similar to a for/next loop in `BASIC`.

NOTES When ToolBook has completely executed the `step` control structure, the value of `<variable>` is the value of `<finish>` plus the value of `<steps>`. For example, if `<finish>` is `10` and `<steps>` is `1`, then `<variable>` is `11`.

Use the `continue` command to immediately force the next iteration of the control structure, and use `break` to exit the control structure before the `variable` has reached the `finish` value.

The `<start>`, `<finish>`, and `<step>` parameters are evaluated only once each time ToolBook executes the `step` control structure.

PARAMETERS

PARAMETER	DESCRIPTION
`<variable>`	A variable name.
`<start>`	Expressions that evaluate to a number.
`<finish>`	Expressions that evaluate to a number.
`<steps>`	A number. If not specified, the default is 1. If the value is positive, the variable is incremented by this number until its value is greater than or equal to the value of `<finish>`. If the value is negative, the variable is decremented by that number until its value is less than or equal to the value of `<finish>`.
`<statements>`	Any OpenScript statements to be repeated.

EXAMPLES

```
step k from 1 to 10
    request k
end
```

```
x = 10
y = 20
step ctr from x to y by 2
    request ctr
end
```

```
-- remove all "even" numbers from a list of numbers
x = "1,2,3,4,5,6,7,8,9,10,13,16,17,18,21,29,44"
step j from itemCount(x) to 1 by -1
    curVal = item j of x
    if curVal mod 2 = 0
        clear item j of x
    end
end
request x
```

storedImages

Property

DESCRIPTION The background or page property that contains information about the stored images available for various display devices.

NOTES This is a read only property and cannot be set.

VALUE The returned value is either `null` or a series of textlines where each textline contains the following information in this format:

```
<bits>,<color planes>,<x units>,<y units>,<size>
```

PARAMETER	DESCRIPTION
`<bits>`	The number of bits per `pixel` used by the display device.
`<color planes>`	The number of color planes used by the display device.
`<x units>`	The number of logical horizontal units used by the display device.
`<y units>`	The number of logical vertical units used by the display device.
`<size>`	The size of the stored image in kilobytes.

EXAMPLES

```
sz = item 5 of storedImages of this page
```

```
if item 1 of storedImages of this background <> 24
    request "Not 24 bit image"
end
```

storeImage

Property

DESCRIPTION A background or page property that specifies whether the offscreen image of the background or page is stored automatically. You can get or set this property.

The stored image is device-dependent, so different images can be stored in the book's file for VGA and Super VGA, for example. You can store images for more than one kind of system by running the book and storing images on each system. ToolBook automatically uses the appropriate stored image for the current system. You can see a list of the stored images in the Page Properties or Background Properties dialog box.

Do not store page or background images until the final step of optimization. For relatively simple backgrounds and for pages that display information in recordfields, set `storeImage` to `true` for the background, set `drawDirect` to `false` for background objects, and set `drawTextDirect` to `true` for recordfields. However, on low-memory systems, you may get better results by setting `storeImage` to `false` for the background and `drawDirect` to `true` for its objects.

NOTES You can get or set this property.

VALUE `True` or `false`; the default is `false`.

If `storeImage` is set to `true`, a compressed image of the page or background is saved each time you leave the page or background; the image is not updated when a user navigates to another page.

If `storeImage` is set to `false`, all stored images are removed for the current page or background.

Setting `storeImage` to `true` is the same as checking the Store Image option in the Background Properties or Page Properties dialog box.

EXAMPLES

```
storeImage of this page = storeImage of this background
```

```
storeImage of this page = x
```

```
get storeImage of this background
```

stretchGraphic

Property

DESCRIPTION A button property that specifies whether the assigned graphic resource is stretched to fill the entire graphic area of the button.

NOTES The opposite effect of `stretchGraphic` would be an auto size option to allow the button to automatically resize itself to the size of the assigned graphic resource.

You will find that feature under the user property entry `ASYMI_AutoSize`.

VALUES `True` or `false`.

The default is `false`.

If `stretchGraphic` is true, all bitmap graphics for any button state are stretched to fill the entire face of the button.

EXAMPLES

```
stretchGraphic of button "TR44" = true
```

strikeout

Character Formatting

DESCRIPTION Sent to the current page when `strikeout` is chosen from the `Text` menu.

You can also `send` this message using the `send strikeout` statement. ToolBook's default response is to change the type style to strikeout or, if it's already strikeout, to turn off that style.

EXAMPLES

```
select chars 15 to 20 of text of field "list"
send strikeout
```

strokeColor

Property

DESCRIPTION A property of a background, draw object, recordfield, field, button, combobox or graphic object that specifies the object's stroke color in `HLS` values.

NOTES This property can be used interchangeably with `rgbStroke`, but its values are expressed in `HLS` values instead of `RGB` values. When you change the setting for `strokeColor`, the setting for `rgbStroke` changes as well.

VALUES A list of three non-negative numbers representing `hue`, `lightness`, and `saturation`, or a valid `color constant`.

The default is the value of `sysStrokeColor` when the object was created.

The valid range for hue is `0` to `360`, for lightness, `0` to `100`, and for saturation, `0` to `100`.

EXAMPLES

```
strokeColor of button "Apple" = red
strokeColor of button "Apple" = 20,20,50
strokeColor of button "Apple" = strokeColor of button "Orange"
```

studentName

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of a book that holds the name of the current student who launched the content via an LMS.

NOTES You can get but not set this property in OpenScript as well as in the Actions Editor.

In OpenScript, this can also be achieved by calling the `ASYM_CMS_UserName()` function.

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

VALUES `True` if the page has been visited before, otherwise `false`.

EXAMPLES

```
text of field "FullName" = studentName of this book
```

style

Property

DESCRIPTION A property of a viewer that specifies a list of style options that define the elements included in the frame of a viewer. Elements include the minimize box, maximize box, Control menu, and scroll bars. The list can contain one or more style options.

A property of the tool bar that specifies whether the tool bar is displayed with graphics only, captions only, or both graphics and captions simultaneously.

NOTES You can get or set this property.

VALUES For `viewers`, a list of `0` or more of the following values: `maxBox`, `minBox`, `scrolling`, `sysMenu` or `null`.

The default is "`scrolling,maxBox,minBox,sysMenu`".

For the `tool bar`, a list of one or more of the following values: `graphics`, `captions`, or a list of both values ("`graphics,captions`").

The default is the current value of `style=` in the `[Palettes]` section of the `INSTRUCTOR.INI` file.

If no value is found in `INSTRUCTOR.INI` file, the default is `graphics`.

EXAMPLES

```
style of viewer "help" = null
```

```
style of viewer "form" = "sysMenu,minBox"
```

subscript

Character Formatting

DESCRIPTION Sent to the current page when `subscript` is chosen from the `Text` menu.

You can also `send` this message using the `send subscript` statement. ToolBook's default response is to change the type style to subscript or, if it's already subscript, to turn off that style.

EXAMPLES

```
select chars 15 to 20 of text of field "list"
send subscript
```

sum()

Statistical Values

SYNTAX `sum(<list>)`

DESCRIPTION Returns the sum of a list of numbers.

PARAMETERS

PARAMETER	DESCRIPTION
`<list>`	An expression that results in a list of numbers.

EXAMPLES

```
lst = sum(14,35,66,85,940)  -- results in 1140
```

```
-- determine total character length of all page names
x = null
step k from 1 to pageCount of this book
    push charCount(name of page k) onto x
end
answ = sum(x)
```

superscript

Character Formatting

DESCRIPTION Sent to the current page when `superscript` is chosen from the `Text` menu.

You can also `send` this message using the `send superscript` statement. ToolBook's default response is to change the type style to superscript or, if it's already superscript, to turn off that style.

EXAMPLES

```
select chars 15 to 20 of text of field "list"
send superscript
```

suspendMessages

Property

DESCRIPTION Prevents a control from generating event messages.

WARNING THIS IS AN UNDOCUMENTED FEATURE. What that means is that you can use it but you will most likely be unable to get assistance from Click2learn on how to use it, or troubleshoot it.

NOTES This tells the ActiveX control not to generate any events. This can improve performance in a case where you're not interested in an ActiveX controls events.

VALUES `True` or `false`, the default is `false`.

EXAMPLES

```
suspendMessages of Tlist "mainlist" = true
```

syd()

Financial Values

SYNTAX `syd(<cost>,<salvage>,<life>,<period>)`

DESCRIPTION A financial function that returns the depreciation of an asset for a specified period using an accelerated depreciation method. This function is similar to `ddb()`, but `syd()` distributes a greater amount of depreciation through the earlier years of the asset's life, rather than placing a greater amount of the depreciation in the first year.

PARAMETERS

PARAMETER	DESCRIPTION
`<cost>`	A positive number representing the initial cost of an asset.
`<salvage>`	A positive number representing the value of the asset at the end of its life expectancy (sometimes called the salvage value of an asset).
`<life>`	A positive whole number representing the number of periods in the life expectancy of the asset.
`<period>`	A positive whole number representing the period for which the depreciation is calculated (`<period>` must use the same unit as `<life>`).

EXAMPLES

```
-- Calculates the depreciation allowance of a new car
x = syd(10500,1000,10,10)
```

sys3DInterface

System Property

DESCRIPTION A system property that specifies whether all dialog boxes opened in ToolBook are displayed with three-dimensional controls.

NOTES You can get or set this property.

This property affects the appearance of Windows dialog boxes and built-in ToolBook dialog boxes.

You can change the default by setting `startup3DInterface` to false.

VALUES `True` or `false`.

The default is `true`.

If `true`, all dialog boxes that appear in ToolBook are drawn with three-dimensional controls.

If `false`, all dialog boxes are drawn using the default Windows style.

sysAlignment

System Property

DESCRIPTION A system property that specifies the default `textAlignment` to be used for text in newly created fields and recordfields.

sysBooks

System Property

DESCRIPTION A system property that specifies the books to be used as system books.

NOTES You can get or set this property.

ToolBook searches for `handlers` in the book scripts of system books after it searches the current book's script. The system books are searched in the order listed in `sysBooks`.

VALUES A list of ToolBook file names, with path names if necessary.

The default is `null`.

sysCentered

System Property

DESCRIPTION A system property that specifies the default setting for the `Draw Centered` option.

NOTES You can get or set this property.

Setting this value is the same as choosing `Draw Centered` from the `Draw` menu.

VALUES `True` or `false`.

The default is `false`.

If `true`, any object drawn in the book is drawn from its center.

sysChangesDB

System Property

DESCRIPTION A system property that specifies whether ToolBook displays the `Save Changes` dialog box when a user closes a book in which changes have been made.

NOTES You can get or set this property.

When `sysChangesDB` is `true`, ToolBook displays the Save Changes dialog box.

If you need additional control over what conditions display the Save Changes dialog box, use the `saveOnClose` property in conjunction with `sysChangesDB`.

VALUES `True` or `false`.

The default is `true`.

If `true`, ToolBook displays the Save Changes dialog box when a user closes a book, if changes were made.

sysClientHandle

System Property

DESCRIPTION A system property used to get the `16-bit` window handle of the client window in the Main window.

This is equivalent to `clientHandle of mainWindow`.

NOTES This property cannot be set.

The Main window's client window is the area where objects are created and displayed. The client window does not include the title bar, menu bar, or window borders.

VALUES The `16-bit` window handle of the client window, which is assigned by Windows.

sysClientHandle32

System Property

DESCRIPTION A system property used to get the `32-bit` window handle of the client window in the Main window.

This is equivalent to `clientHandle32 of mainWindow`.

NOTES This property cannot be set.

The Main window's client window is the area where objects are created and displayed. The client window does not include the title bar, menu bar, or window borders.

VALUES The `32-bit` window handle of the client window, which is assigned by Windows.

sysCommandLine

System Property

DESCRIPTION Returns the command line that was used to start the ToolBook instance.

NOTES You can get but you cannot set this property.

VALUES This system property contains the same value returned in the command line parameter of the `enterSystem` handler.

sysCursor

System Property

DESCRIPTION A system property that specifies the shape of the cursor that appears onscreen.

NOTES You can get or set this property.

If you change the value of `sysCursor`, that value remains in effect until the value is changed again or until the end of the ToolBook session.

The `default` shape results in standard ToolBook cursor behavior. For example, the cursor changes automatically to indicate hotwords in fields.

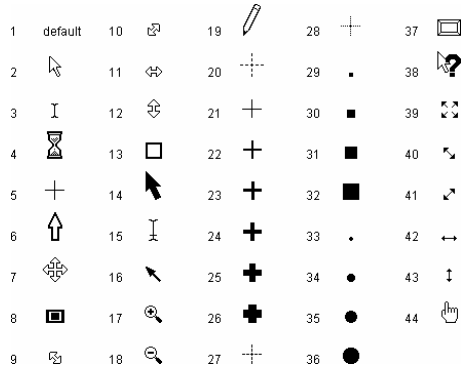
VALUES `None` (for no cursor), `default`, an integer from `1` to `44`, or a cursor resource reference.

Cursors `2-12` are standard Microsoft Windows cursors.

Cursors `7` and `8` are not available with Windows 95 and 98 and NT 4, so ToolBook automatically sets them to cursor `2`.

Cursors `9-12` are Windows 3.1 cursors. In Windows 95 and 98 and NT 4, they appear as cursors `40-43`.

Cursors `13-44` are ToolBook cursors. Because they are standard Windows cursors, they are system dependant.



EXAMPLES

```
sysCursor = 44
sysCursor = "none"
sysCursor = cursor "leftHandPointer"
```

sysDate

System Property

DESCRIPTION A system property used to get the current system date, which is set by the computer.

NOTES This property cannot be set.

VALUES The current system date in the format specified by `sysDateFormat`.

sysDateFormat

System Property

DESCRIPTION A system property that defines the format of `sysDate` and how dates are formatted by the `format` command.

NOTES You can get or set this property.

The value of this property only affects the value of `sysDate`, unless you also use the `format` command to format the date.

To use a date in an arithmetic calculation, format the date as `"seconds"`.

VALUES A string of placeholder characters enclosed in quotation marks.

The default is based on the values of `sysEvening`, `sysMorning`, and `sysShortDate`, which are based on Windows configuration settings.

For a table of the placeholder characters, see the entry for `format`.

EXAMPLES

```
sysDateFormat = "mm/dd/yy"
sysDateFormat = "seconds"
```

sysDrawDirect

System Property

DESCRIPTION A system property that specifies the default value of `drawDirect` for new objects.

sysError

System Property

DESCRIPTION A system property that specifies the nature of a `nonfatal` error (can also include fatal errors if `sysSuspend` is set to false).

NOTES A `nonfatal` error is an error that, although it may have consequences in script execution, does not cause ToolBook to open the `Execution Suspended` message. When a nonfatal error occurs, ToolBook sets this property but takes no other action.

Several commands set `sysError` when they detect an unusual or exceptional condition. For example, createFile sets this property to `read only` when an attempt is made to create a file that already exists and has the read-only attribute. ToolBook also sets `sysError` when `sysSuspend` is set to false and an error occurs that would normally cause ToolBook to open the `Execution Suspended` message.

Set `sysError` to `null` before beginning an operation in which you will check and use the value of `sysError`.

VALUES A string that describes the cause of the last error.

The default is `null`.

Each `sysError` string has a corresponding error number. For a list of `sysError` strings and corresponding error numbers, see the `sysErrorNumber` property.

EXAMPLES

```
-- User-defined function that creates a new file
-- and validates that it was created
to get createNewFile pFileName
    clear sysError
    sysSuspend = false
    createFile pFileName
    sysSuspend = true
    if sysError = null
        return true
    end
    return false
end
```

DESCRIPTION A system property that specifies the number corresponding to the value of sysError.

NOTES This property cannot be set.

This property provides a language-independent method of determining which error occurred.

VALUES The default is `0`, which corresponds to a `null` value for `sysError`.

Below is a list of `sysErrorNumbers` and corresponding `sysError` values.

VALUE	SYSERRORNUMBER ERROR MESSAGES
1	Internal Error - Bad class
2	The requested page is not in this book.
3	The requested file is not loadable by this application.
4	Internal Error - Bad header offset.
5	Internal Error - Bad OF.
6	Internal Error - Bad object handle.
7	Internal Error - Bad object offset.
8	Internal Error - Bad object pointer.
9	This book was created with a version subsequent to ToolBook 1.5.
10	Not enough room on disk. Delete file(s) and try again.
11	Input or output error reading from or writing to a file.
12	The book is full. No more pages can be added. Remove some pages and try again.
13	Not enough memory. Close other applications or save this book and try again.
14	Internal Error - Initialization of DLL failed.
15	Not enough local memory. Save this book and try again. If failure continues, close this instance.
16	Page or background is full. Try saving this book to a different name, or deleting some objects.
17	Cannot save a book without a name.
18	Internal Error - Buffer supplied is too small.
19	Internal Error - CDB bad status.
20	Internal Error - There are threads still open.
21	There are too many books to continue with this operation.
22	There are too many books to continue with this operation.
23	There are too many books to continue with this operation.
24	There are too many books to continue with this operation.
25	Internal Error - Bad binary file.
26	Internal Error - Bad image.
27	Internal Error - Bad instance handle.
28	Internal Error - Bad thread handle.
29	Internal Error - Bad CDBID.
30	This object is in use and cannot be removed.
31	Internal Error - Bad page.
32	Internal Error - Maximum lock count reached.
33	Windows system failure (GDI). Exit Windows as soon as possible.
34	This file is a read-only file in DOS. Save under another name.
41	The requested window cannot be identified.
43	That file already exists. Replace it?
44	Cannot perform requested operation, object in use.
45	Internal Error - Cannot draw selection handle.
46	There is not enough room on the temp drive to complete this operation.
49	A printer driver is not installed. Run the Control Panel to install a printer.
50	Internal Error - OpenClipboard failed.
51	Internal Error - Read from Clipboard failed.
52	Internal Error - Write to Clipboard failed.
53	Internal Error - EmptyClipboard failed.
54	Internal Error - Bad value: %d.
55	Internal Error - context requires that selection count is 1.
56	Number is outside the valid range for this command.
57	Windows system failure (MakeProcInstance). Exit Windows as soon as possible.
59	This object is protected.

VALUE	SYSERRORNUMBER ERROR MESSAGES
60	Error while reading from the Clipboard. Cannot continue operation.
61	Error while writing to the Clipboard. Cannot continue operation.
62	Internal Error - Selection is bad. Clear the selection before continuing.
64	A problem has occurred during printing. Check the printer and try again.
65	Out of instance memory. Close this book and try again.
66	Object does not exist.
67	Internal Error - No selection to operate on.
68	Internal Error - A hotword is selected at the wrong time.
69	The maximum number of hotwords allowed in a field is 255.
70	Internal Error - No hotwords found in this field.
71	Internal Error - The hotword handle is bad.
72	Not enough Windows GDI memory for this operation.
73	File not found. Try using a complete pathname.
74	Too many files are open. Try increasing the number of FILES in your CONFIG.SYS.
75	Internal Error - Cannot access this file or object.
76	Internal Error - Unexpected end of file.
77	A file with that name is currently in use. Try a different name.
78	Internal Error - Cancel operation.
79	Internal Error - Programming language.
80	Not enough memory to run.
81	The requested Windows application cannot be run.
82	The requested application is not Windows-compatible.
83	Cannot remove the only page in a book.
84	Supply some text to search for.
86	Search completed. Text not found.
87	There are too many objects on this page. Delete objects or make a new page.
88	No fixed field lengths were supplied.
89	No delimiter was supplied.
91	The maximum number of characters allowed in a combobox drop-down listbox is 32,000.
92	This page contains no recordfields.
94	Internal Error - Script is bad.
95	Invalid resource type.
101	That file is currently in use as a temporary file. Use another name.
102	Unable to create a temporary file. Exit Windows and delete files in your TEMP directory.
103	Not a valid where clause.
105	Cannot merge a book into itself.
106	Groups cannot be nested more than 16 levels deep.
107	Not a valid number format.
108	Internal Error - message sent with no focus.
109	Object has been deleted.
110	Invalid delimiter: %s
111	Cannot create an object with no size. Specify a positive width and height.
112	This operation cannot be performed on this object.
113	This operation cannot be performed on a hidden object.
114	The maximum number of characters allowed in a field is 32,000.
115	The maximum number of lines allowed in a field is 5,000.
116	This book has been damaged or there is an internal error.
117	Internal Error - programming language.
119	This network book cannot be opened. Either another user is using it, or you do not have proper privileges for its network location. Try making the book read-only.
120	Failed initializing the network book manager, no network book access available. Make sure the file "TBKNET_BASENAME".EXE is installed in a directory in your path.
121	Internal Error - Old type
122	Storage space for large objects is full. Delete large paintObjects or pictures and try again.
123	Width or height of bitmap exceeds maximum. Cannot be pasted or imported.
124	Not enough memory to create page of requested size. Make page size smaller or try running in enhanced mode.
125	Internal error - invalid icon.
126	Scaling factor of the printing device is ignored.
127	Internal error - property not supported by object.
128	Error occurred in RDLIB, so call rdLastError().
129	Only one instance of a book may be open at a time in the Evaluation Edition.

VALUE	SYSERRORNUMBER ERROR MESSAGES
130	Error during an operation on a linked or embedded object.
131	The object must be positioned inside the section's client area.
132	Named resource already exists.
133	A section of this type already exists.
134	There are too many resources in this book. Delete some or create a new book.
135	No resource import filter available. Check that you have the proper filter installed.
136	The action cannot be completed because the object server is busy. Switch to the object server and correct the problem.
137	The action could not be completed because the object is neither a linked nor embedded object.
139	The current operation involving an OLE object cannot be completed because the object server failed to create a new document.
140	The current operation involving an OLE object cannot be completed because the object server failed to create an embedded object.
141	The current operation involving an OLE object cannot be completed because the object server failed to execute.
142	The current operation involving an OLE object failed because the object server failed to terminate correctly.
143	The update of the selected object failed.
144	The object server for the selected object failed to open the object. Possibly the object link is broken.
145	Communication with the object server for the selected object failed. Possibly the object link is broken.
146	Failed to establish a connection to the object source on a network device. Possibly the object link is broken.
147	The object server for the selected object failed to show itself. Possibly the object link is broken.
148	The object server for the selected object failed to execute the requested action on the object. Possibly the object link is broken.
149	The object server for the selected object does not allow the object to be renamed.
150	The object server for the selected object can not draw the object with the selected format. Try to create the object with another format.
151	The object server for the selected object can not draw the object with the selected format. Try to create the object with another format.
152	There are open objects that may need updating in %s. Proceeding in any case will close the objects.\n\nDo you want to update the open objects before closing them?
153	Import error. Cannot find specified resource.
154	Error on attempt to load a metafile.
155	The operation could not be completed because the OLE object is missing a critical piece of information.
156	The total number of series in a chart may not exceed 99.
157	Not a valid resource.
158	This book is being used by another instance of Calvin.
160	Error on attempt to load a bitmap.
161	The supplied text must match an item in the list.
162	This object cannot be duplicated.
163	Attempted group or ungroup operation invalid in this context.
164	Bad character for input mask or field is full.
165	Failed validation script for focus object.
166	Database column requires entry.
167	Database column must be filled.
168	Entry failed format validity.
169	New data could not be saved to table.
170	Entry value exceeds maximum range.
171	Entry value less than minimum range.
172	Improper script for validation.
173	Entry must match item in list.
174	Looping calculated field references.
175	Expression for calculated field failed.
176	Calculated field chain nested too deep.
177	Expression contains too many references to other objects.
178	Cannot size section to its children.
179	All objects must share one parent for this operation.

VALUE	SYSERRORNUMBER ERROR MESSAGES
180	Exceeded maximum number of characters for column.
181	Unknown OLE server.
182	This book was created with a previous version of ToolBook and cannot be loaded.
183	Invalid OLE object type.
184	Invalid OLE action.
189	Support for E-Mail not available.
191	An editor for the specified resource type is not found in the Resource Editors section of ToolBook's .INI file.
192	Unable to launch editor for the specified resource type.
193	Error occurred while attempting to send mail.
194	Error occurred while attempting to login into mail.
195	User cancelled send mail request.
196	Error occurred while attempting to open mail attachment.
197	Maximum size can not be less than the minimum size.
198	Minimum size can not be greater than the maximum size.
199	The selected font file is tagged as not redistributable and cannot be imported.
200	The selected font file is already embedded.
201	Cannot set focus to requested object.
202	This operation is not supported on this type of resource.
203	This operation can only be performed in an open window.
204	The media resource is not open. Try using \"mmOpen\" before this command.
205	The media resource is already open. Try using \"mmClose\" before this command.
206	The media reference could not be found. Check the reference or file name.
207	The multimedia driver could not open this media. The appropriate drivers may not be available.
208	This is not a media clip: %s.
209	Media may only be played in mmcontainer objects.
210	This media type is not visual, so does not have visual properties.
211	This media type is not audible, so does not have volume properties.
212	The time string is invalid for this time format.
213	This is not a valid time format.
214	This time format is not supported by this media type.
215	This is not a valid media state.
216	The parameter is out of range for this media resource.
217	This media type does not have an external alias.
218	This media type does not support track information.
219	An unexpected multimedia error has occurred.
255	False Alarm. Harmless internal error while returning to top-level.
4001	Not a valid file name.
4002	Cannot find file %s.
4003	That password is not correct.
4004	Name contains an invalid character.
4005	Cannot find the Help book.
4006	Removing this page will also remove its background from the book and the text of any recordfields. Continue?
4007	Passwords do not match.
4008	The selection includes one or more recordfields. Deletion will remove the text of recordfields on all pages. Continue?
4009	The script already has a buttonClick handler. Do you wish to replace it?
4010	Not a valid number.
4013	Ungrouping will destroy the group's script and its name. Continue?
4014	Memory is low. Close books and applications.
4015	Memory is too low to continue. Close books and applications.
4016	This object's properties are already being modified by another instance.
4017	This page's properties are already being modified by another instance.
4018	This background's properties are already being modified by another instance.
4019	This book's properties are already being modified by another instance.
4020	Obsolete error.
4021	Obsolete error.
4022	Please specify a font in the combo-box.
4023	Please specify a point size in the combo-box.
4024	Unable to load requested cursor.

VALUE	SYSERRORNUMBER ERROR MESSAGES
4025	The script is in an old format. It cannot run.
4026	The script has been damaged. It cannot run.
4027	Not enough memory to display palette or paint object: %u.
4028	Not enough memory to display window: %s.
4029	LoadString failed with ID: %u.
4030	Cannot print or access information about the printer.
4031	The DEVICE= line in your WIN.INI is not correct. Run the Control Panel to install a printer.
4032	Cannot load the printer driver: %s.
4033	Bad syntax in the specified conditions.
4034	No pages match the specified conditions.
4035	Cannot start printing. Check the printer and try again.
4036	The current printer cannot print bitmaps.
4037	A problem has occurred during printing. Check the printer and try again.
4038	Cannot start without a book. A book must be specified.
4039	The Clipboard contains too many objects to paste.
4040	Initialization failed. Not enough memory or the file is damaged.
4041	The password contains invalid characters.
4042	The text of this object's script has been removed. You cannot edit it.
4043	The object whose script you are editing has been deleted.
4044	Point size must be a positive integer.
4045	The help file for this context cannot be found in your path.
4046	Cannot run this version with older versions of ToolBook DLLs.
7001	Save current changes to: %s?
7002	Cannot save to a file name with wildcard characters.
7003	File %s already exists. Replace it?
7004	Warning: Windows is overloaded. You cannot scroll by dragging outside the field bounds.
7005	You must save the book before you can link.
7006	You cannot remove the only page in a book.
7007	Warning: This operation cannot be reversed using Undo.
7008	One or more of the specified margins lies outside the printable page area. Keep specified margins?
7009	You cannot link to a background. Go to the foreground and continue.
7010	Warning: Edits to the Color Tray will be lost if you continue.
7011	One of the link connections has been destroyed. The link cannot be completed.
7012	An unrecoverable error in file operations has occurred (possibly due to low memory). A new file has been started.
7013	Warning: '%s'is a read-only file in DOS. Changes cannot be saved.
7014	This version of the application requires Microsoft Windows(tm) version 3.
7015	Cannot find the most recent page. You have been moved to Page 1.
7016	Cannot complete the link operation. The page may be full or memory may be low.
7017	Low memory.
7018	This book was created with version ToolBook 1.0. If you save changes to it, you will be unable to open it with version 1.0. Do you want to save changes to it?
7019	This book was created with version ToolBook 1.0. If you save changes to it, you will be unable to open it with version 1.0. Do you want to save changes to: %s?
7021	Low on system memory. Draw direct objects may not draw correctly when magnified.
7022	Low on system memory. Draw direct objects may not draw correctly in print preview.
7023	Book %s can no longer be accessed at author level with the ToolBook Evaluation Edition.
7024	Evaluation Edition allows 6 hours (360 minutes) at Author level for each book.\r\n\r\nTime remaining for %s: %u minutes %u seconds.
7025	This bitmap format is not supported.
7026	No bitmap is on the Clipboard.
7027	Error opening file.
7028	File type not recognized.
7046	Warning: Query edits will be lost if you continue.
7047	Warning: Schema edits will be lost if you continue.
8000	Internal error: Unknown PL error code
8001	Not enough memory to compile. Try closing other instances or applications.
8002	Not enough memory to compile. Try closing other applications.
8003	Not enough memory to compile. Try quitting and restarting ToolBook.
8004	Internal error: PL module caller.

VALUE	SYSERRORNUMBER ERROR MESSAGES
8005	Internal error: PL module.
8006	Syntax error.
8007	Unterminated string.
8008	Internal error: Compiler stack overflow.
8009	Internal error: Unknown compiler error.
8010	Internal error: Feature not yet implemented.
8011	Cannot find page number %ld.
8012	Cannot find page \"%.100s\".
8013	Variable %.100s has no value.
8014	Cannot perform requested operation on a null string.
8015	Not a selectable string specifier: %.100s.
8016	Not a selectable object: %.100s.
8017	Not a valid OpenScript term: %.100s.
8018	Outside the range 0 to 65535: %.100s.
8019	This item does not have properties: %.100s.
8020	Maximum menu length %d exceeded by menu name \"%.80s\".
8021	Cannot execute this command in runtime version.
8022	%.100s
8023	Cannot find background \"%.100s\".
8025	Value not a page.
8026	Cannot find background ID %ld.
8027	Not a valid color: %.100s.
8028	Not a stack: %.100s.
8029	No handler for message %.100s.
8030	Random(%ld) is outside the range of 1 to 32767.
8031	%.100s error in %.80s(%g,%g)
8032	There is no object named %.100s.
8033	Property not supported by object: %.100s.
8034	Cannot set object property as requested.
8035	Cannot get object property as requested.
8036	Cannot paint.
8037	Not a valid draw tool.
8040	Not a graphic object: %.100s.
8041	Not a page: %.100s.
8042	Not a valid list of four numbers: %.100s.
8043	Wrong number of vertices for this object.
8044	Cannot set this system property. Check that it is settable.
8045	Cannot get system property.
8047	Value \"%.100s\" is not valid in this context.
8048	There is no selection to act on.
8049	Not a time: %.100s.
8050	Not a date: %.100s.
8051	Cannot find file: %.100s.
8052	Cannot create or write to read-only file %.100s.
8053	Too many files are open. Try closing one or more files.
8054	Error when writing to file %.100s.
8055	Error when reading from file %.100s.
8057	Not a valid file name: %.100s.
8058	Not enough memory. Try quitting and restarting ToolBook.
8059	Sharing violation. File already in use.
8060	No such file.
8061	Read only.
8062	***undef***.
8063	Too many files are open. Try closing one or more files.
8064	I/O error.
8065	File read/write error.
8066	End of file.
8067	Cancel
8068	Error when starting spooler.
8069	Can set text property only for fields.
8070	Cannot run %.100s.
8071	Magnification power must be 1, 2, 4, 8 or 16.

VALUE	SYSERRORNUMBER ERROR MESSAGES
8072	Not enough memory to run %.100s. Try closing other instances or applications.
8073	Trace stop in
8074	Breakpoint in
8075	Can include Return statement only in To Get handlers.
8077	Cannot get argument %d. Handler has only %d arguments.
8078	\"%.100s\" is not a window or an object.
8079	Not a container: %.100s.
8080	%d - %.60s in %.100s.
8081	Invoked for %.100s.
8082	Script for %.100s.
8083	ToolBook Debugger
8087	You cannot edit the script of a protected object.
8089	Object \"%.100s\" is not valid in this context or has been deleted.
8090	Error when compiling script: %.100s.
8091	Must be at Author level to execute requested command.
8093	Unable to format date.
8094	Unable to format time.
8095	Value \"%.100s\" is out of range.
8097	Command Window
8098	Cannot pop from empty stack.
8099	No menu item named \"%.100s\".
8100	Not enough memory to change menu as requested. Try closing other instances or applications.
8101	There is no menu item at position %d of column %d.
8103	Cannot have more than 60 menus and 255 items per menu.
8104	Menu already exists: \"%.100s\".
8106	No menu named \"%.100s\".
8108	Cannot load DLL \"%.100s\".
8110	Error: %.100s.
8112	No search to repeat.
8113	Cannot set property of a protected object: %.100s.
8115	Script interrupted by user in
8116	Too many objects selected for the requested operation.
8118	Calls to handlers are nested too deeply. Try making fewer nested calls.
8119	Continue search at beginning?
8120	Print command must be within a Start Spooler control structure.
8121	Cannot send message because cannot find target %.100s.
8123	OK
8124	Failed: Denied
8125	Failed: Busy
8126	Failed: Memory Error
8127	Failed: No Server
8128	Failed: Interrupted
8129	%.100s,%d,%d,%d,%d,%d,%d,%d,%d
8131	No DDE request to respond to.
8132	Cannot sort while in background. Switch to foreground.
8134	Not a book or background: %.100s.
8135	Not a book or page: %.100s.
8136	Not a logical value (must be True or False): %.100s.
8137	Not a valid type of alignment: %.100s.
8138	Not valid parameter (must be Fixed or Delimited): %.100s.
8139	Not a string: %.100s.
8140	Outside the range -2147483648 to 2147483647: %.100s.
8141	Outside the range 1 to 2147483647: %.100s.
8142	Outside the range -32768 to 32767: %.100s.
8143	Outside the range 0 to 32767: %.100s.
8144	Not a number: %.100s.
8145	Not a valid string specifier: %.100s.
8146	Not a valid list of two numbers: %.100s.
8147	Not a book, page or background: %.100s.
8148	Not a window or movable object: %.100s.
8149	Not a page or background: %.100s.

VALUE	SYSERRORNUMBER ERROR MESSAGES
8150	Not a book: %.100s.
8151	Not an object: %.100s.
8152	The object with this ID is not a(n) %.100s.
8153	Outside the range 1 to 65535: %.100s.
8154	File \"%.100s\" not open. Open file first.
8155	Cannot find system book \"%.100s\".
8159	No recordfields found to sort.
8161	Record too large.
8162	Not a valid object reference.
8163	Cannot select a background.
8164	Cannot select text of \"%.100s\". It is not a field or recordfield.
8165	Can only select objects on current page or background.
8166	Cannot nest Start Spooler commands.
8167	End Spooler cannot find \"%.100s\".
8168	DLL \"%.100s\" is not loaded.
8169	No function named \"%.50s\" in DLL \"%.100s\".
8170	No function at ordinal %d in DLL \"%.100s\".
8172	Wrong type of argument for DLL function.
8174	Cannot set printer property.
8175	Object too small to draw.
8176	Cannot make requested selection.
8177	Cannot save changes to an untitled book. Try using Save As command.
8178	Cannot execute menu command as requested.
8179	Cannot paste in this context. Clipboard does not contain text.
8180	Undo is not available in this context.
8181	Repeated [type] in FILL, each must be different.
8182	Cannot perform desired Clipboard operation.
8183	Cannot create hotword as requested.
8184	Cannot reshape selected object.
8185	DLL function returned 0L.
8186	Cannot find page ID %ld.
8187	Cannot start recording as requested.
8189	Missing or invalid recordfield name.
8190	Outside the range 0 to 32767: %d.
8191	The resulting string is too long.
8192	The script for this object has been modified.
8194	Menu identifier %.100s is not a valid menu or menu item name.
8195	Missing or incorrect focus in Search Again.
8196	Not a valid pattern: %.100s.
8197	Search already in progress...
8198	Not a valid search parameter when on the background.
8199	Outside the range 1 to 32767: %.100s.
8200	Not a valid window handle: %d
8201	Not a recordfield on the current page: %.100s
8202	The valid range for sysCursor is 1 to 38: %d is outside this range.
8203	Unable to load filter %s. File does not exist or is not a valid DLL.
8204	%s is not a valid graphics import filter.
8205	Unable to import %s. File type not supported by available filter.
8206	Unable to import %s. File type not supported by available filters.
8207	Unable to import %s. Unsupported bitmap file format.
8208	Unable to import %s. File does not contain a picture.
8209	%s is not a Windows 3.0 metafile.
8210	Warning: The bitmap in this file may be corrupted.
8211	Storage space for large objects is full. Delete large paintObjects or pictures and try again.
8212	Width or height of bitmap exceeds maximum. Cannot be pasted or imported.
8213	Can't contain other objects: %.100s.
8214	Specified file is not a valid icon file: %.100s.
8215	Outside the range 0 to 4294967295: %.100s.
8216	Not a valid color table: %.100s.
8217	Specified file is not a valid icon file: %.100s.
8218	Not a valid font style: %.100s.
8219	Not a valid border style: %.100s.

VALUE	SYSERRORNUMBER ERROR MESSAGES
8220	Not a valid line style: %.100s.
8221	Not a valid line spacing style: %.100s.
8222	Not a valid tab style: %.100s.
8223	Not a valid paragraph style: %.100s.
8224	Not a valid level: %.100s.
8225	Not a valid unit of measurement: %.100s.
8226	Not a valid cursor type: %.100s.
8227	Not a valid window object: %.100s.
8228	Not a valid print style: %.100s.
8229	Not a valid field or recordfield style: %.100s.
8230	Not a valid field, recordfield, or combobox: %.100s.
8231	Not a valid field, recordfield or button: %.100s.
8232	Not a valid field, recordfield or hotword: %.100s.
8233	Object does not support caption property: %.100s.
8234	Not a valid button: %.100s.
8235	Not a valid button or hotword: %.100s.
8236	Object does not support color: %.100s.
8237	Not a valid object class: %.100s.
8238	There is nothing to GO BACK to.
8239	Not a valid menu level specifier.
8240	Not a valid key constant: %.100s.
8241	Not a valid key state, must be UP or DOWN: %.100s.
8242	Not a background: %.100s.
8243	Not a valid list of objects: %.100s.
8244	Not a valid list of points: %.100s.
8245	Too many system variables.
8246	Not enough segment memory to set the object's script.
8247	Object does not support user definable properties: %.100s.
8248	Not a drawable object: %.100s.
8249	Auto-Recompile
8250	Error compiling script in: %s. No part of it will execute. Save it anyway?
8251	Out of memory compiling script in: %s. No part of it will execute. Save it anyway?
8252	Value \"%ld\" is out of range.
8253	Outside the range 1 to 32767: %ld.
8254	Cannot select a book.
8255	Cannot set this system property to the value given.
8256	Not a valid hotword style: %.100s.
8257	Not a valid connection reference: %.100s.
8258	Not a valid query reference: %.100s.
8259	Not a valid value type: %.100s.
8260	With the Evaluation Edition, you cannot edit scripts which are not in the current book, or in books which have expired.
8261	Cannot find connection number %ld.
8262	Cannot find connection \"%.100s\".
8263	Cannot find connection ID %ld.
8264	Cannot find query number %ld.
8265	Cannot find query \"%.100s\".
8266	Cannot find query ID %ld.
8267	Not an array: %.100s.
8268	Not a valid Effect: %.100s.
8269	Not a valid Direction: %.100s.
8270	Not a valid Speed: %.100s.
8271	Cannot initialize array values as requested.
8272	Attempt to access element outside bounds of the array.
8273	Not a valid array fill qualifier: %.100s.
8274	Command or expression not supported in this version of ToolBook.
8275	Attempt to redefine array.
8276	Not a valid chart style: %.100s.
8277	Zeros passed for pmt and rate parameters of nper function.
8278	Zeros passed for nper and rate parameters of pmt function.
8279	Not a valid style: \"%.100s\".
8281	Not a valid type: \"%.100s\".

VALUE	SYSERRORNUMBER ERROR MESSAGES
8282	Not a valid state: \"%.100s\".
8283	Not a valid list of window styles: \"%.100s\".
8284	Not a valid parent window: %.100s.
8285	Nested Reports not allowed %.100s.
8286	Not a valid report page %.100s.
8287	Section too large %.100s.
8288	Invalid section properties for section type %.100s.
8289	Report group order and query dependency are in conflict %.100s.
8290	Only one section type %.30s allowed.
8291	No queries attached to this report page %.100s.
8292	Not a valid setting: \"%.100s\", must be Yes, No, Ask, or System.
8293	Not a valid resource reference: \"%.100s\".
8294	Not a valid query or connection reference \"%.100s\".
8295	Not a valid setting: \"%.100s\", must be Automatic or Manual.
8296	Unable to locate resource manager application.
8297	Not a valid query column reference \"%.100s\".
8298	Not a valid graphic button drawing style: %.100s.
8299	Not a valid drawing tool: %.100s.
8300	Not a valid graphic button border style: %.100s.
8301	Not a valid graphic button caption position: %.100s.
8302	Not a valid resource type: \"%.100s\".
8303	Not a valid SeekFile origin: %.100s.
8304	Unable to locate MMSYSTEM.DLL.
8305	Incompatible version of MMSYSTEM.DLL.
8306	Unable to play \"%s\".
8307	Not a valid system metric variable: %.100s.
8308	Notify handlers are not allowed in objects of this type.
8309	Not a valid line ends style: %.100s.
8310	Not a valid line ends size: %.100s.
8311	Not a valid DLL pointer argument or return type: %.100s.
8312	Array does not match requirements for operation.
8313	Cannot perform requested operation, an object must have the focus.
8314	Not a valid caret location: %.100s.
8315	The text for this script has been stripped.
8316	This object type can not be copied.
8317	Not a valid Cell (Row,Column) address \"%.100s\".
8318	Not a valid Cell Range Specification \"%.100s\".
8319	Not a valid Disk Specification \"%.100s\".
8320	Not a valid Column Parent \"%.100s\".
8321	Not a valid Column reference \"%.100s\".
8322	Cannot clear the main window.
8323	Cannot create a circular parent window reference.
8324	Cannot perform this operation until window is opened.
8325	This borderStyle is not acceptable to this object \"%.100s\".
8326	Hide/Show not supported for object.
8327	Cannot link DLL function: \"%.100s\".
8328	Not found.
8329	Not a page or graphic.
8330	Not a valid tile type: \"%.100s\".
8331	Handled.
8332	Not enough global memory for report generation.
8333	A report must have at least one section.
8334	This section has some invalid properties \"%.100s\".
8335	A nested printing has occurred when printing \"%.100s\".
8336	An internal inconsistency error has occurred.
8337	The calculated total logical page width exceeds physical paper size.
8338	The target paper size and the physical paper sizes differ.
8339	A section of this type do not support this property.
8340	Cannot remove hotword as requested.
8341	Not a valid minimum or maximum size: \"%.100s\".
8342	Not a valid default position: \"%.100s\".
8343	Not a viewer: \"%.100s\".

VALUE	SYSERRORNUMBER ERROR MESSAGES
8344	Not a valid default client size: \"%.100s\".
8345	Not a valid aggregate type: \"%.100s\".
8346	Not a valid aggregate category: \"%.100s\".
8347	Not a valid report expression: \"%.100s\".
8348	Not a valid date/time increment specifier: \"%.100s\".
8349	Not a valid date part: \"%.100s\".
8350	Not a valid time part: \"%.100s\".
8351	Not a valid format data type: \"%.100s\".
8352	Not a valid chunk type: \"%.100s\".
8353	Not a valid return type: \"%.100s\".
8354	Not a valid case-sensitivity specifier: \"%.100s\".
8355	Not a valid date interval specifier: \"%.100s\".
8356	Not a valid built-in icon type: \"%.100s\".
8357	Not a valid keyword in this context: \"%.100\".
8358	Not a valid special effect: \"%.100s\".
8359	Wrong number of arguments supplied to handler. Expected %d, received %d.
8360	Not a valid setting for printer scaling: \"%.100s\".
8361	Not a valid caption bar: \"%.100s\".
8362	The selection is not a single picture object.
8363	Not a dragable object: %.100s.
8364	Spell check has reached the end. Continue from the beginning?
8365	The spell check of the selected text is complete. Do you wish to continue?
8366	The spell check of this object is complete. Do you wish to continue?
8367	The spell check of the selected objects is complete. Do you wish to continue?
8368	The spell check of this page is complete. Do you wish to continue?
8369	Missing or incorrect focus in Spell check.
8370	Spell check already in progress...
8371	Not a valid spell check parameter when on the background.
8372	Spell check completed.
8373	The graphic of operator requires a single character argument.
8374	Not a Bitmap, Icon or Cursor resource reference: \"%.100s\".
8375	Not a valid palette style: %.100s.
8376	Bad parameter passed to spelling module.
8377	General failure in spell checker.
8378	Cannot update .INI file.
8379	Cannot initialize spell checker.
8380	No user dictionary specified.
8381	Illegal to copy this object type to %.100s.
8382	Cannot perform this operation on the Main window.
8383	Default page reference not valid: %.100s.\nDefaulting to page 1.
8384	%.100s is not a dynamic array for RESET statement.
8385	Cannot perform requested operation on an array variable.
8386	Not a valid transition destination - must be page or color: %.100s.
8387	Cannot load language dictionary.
8412	The 32-bit subsystem is unavailable.

sysFillColor

System Property

DESCRIPTION A system property that specifies the default `fillColor` used when creating new objects.

sysFontFace

System Property

DESCRIPTION A system property that specifies the default `fontFace` used when creating new objects.

sysFontSize

System Property

DESCRIPTION A system property that sp ecifies the default `fontSize` used when creating new objects.

sysFontStyle

System Property

DESCRIPTION A system property that specifies the default `fontStyle` used when creating new objects.

sysGifInterlaced

System Property

DESCRIPTION Specifies whether a bitmap resource will have interlacing turned on or off when it is exported as a `GIF` file.

VALUES `True` or `false`.

The default is `false`.

sysGrid

System Property

DESCRIPTION A system property that specifies whether the `grid` is visible.

NOTES You can get or set this property.

Setting this property causes an error in Runtime ToolBook.

VALUES `True` or `false`.

The default is `false` (the grid is hidden).

sysGridSnap

System Property

DESCRIPTION A system property that specifies whether objects snap to the `grid` lines when created, moved, or sized.

NOTES You can get or set this property.

Setting this property causes an error in Runtime ToolBook.

VALUES `True` or `false`.

The default is `false`.

sysGridSpacing

System Property

DESCRIPTION A system property that indicates the horizontal and vertical distance between grid lines.

NOTES You can get or set this property.

Setting this property causes an error in Runtime ToolBook.

VALUES A positive whole number from `30` to `4320` in `page units`.

The default is `180`.

sysHistory

System Property

DESCRIPTION A system property that specifies the items in the history stack.

NOTES You can get or set this property.

This stack contains a list of the unique names of the last `100` pages displayed (in the order they were displayed) in the Main window during the current ToolBook instance.

As ToolBook displays each page, it pushes the `uniqueName` of the page onto `sysHistory`.

ToolBook uses the value of this property to determine the content of the History dialog box.

ToolBook does not provide history stack support in viewers other than the Main window.

VALUES A list of the uniqueNames of the last `100` pages displayed in the current ToolBook session.

sysHistoryRecord

System Property

DESCRIPTION A system property that specifies whether ToolBook adds pages to the history stack in `sysHistory`.

NOTES You can get or set this property.

While `sysHistoryRecord` is set to `false`, any pages that are displayed in the Main window are not added to the history of the book, but the History dialog box can still be displayed.

VALUES `True` or `false`.

The default is `true`, which allows pages to be added to `sysHistory`.

sysHotwordsShown

System Property

DESCRIPTION A system property that specifies whether a hotword is shown on the page.

NOTES You can get or set this property.

When a user chooses `Show Hotwords` from the `Text` menu, the `showHotwords` message is sent to the current page. ToolBook's default response is to toggle the value of the `sysHotwordsShown` property.

VALUES `True` or `false`.

The default is `true`.

When `sysHotwordsShown` is `true`, the hotwords are shown in the current `hotwordStyle` setting for the book.

The default `hotwordStyle` for books is `color`. If `sysHotwordsShown` is `true` and the `hotwordStyle` of the book is `color`, the hotwords are displayed in the `hotwordColor`. The default `hotwordColor` is red.

If `false`, the hotword appears as normal text (its style is not shown).

sysIdleDelay

System Property

DESCRIPTION Controls the amount of time, in milliseconds, ToolBook will `yield` to other applications before sending an `idle` notification message.

VALUES A value representing the desired idle delay in milliseconds.

The default value is `1`.

Setting this property to `-1` will cause the behavior of previous versions of ToolBook by effectively causing ToolBook to generate idle messages as fast as possible.

Setting this property to `0` will cause ToolBook to yield to other processes before generating an idle message.

sysIndents

System Property

DESCRIPTION A system property that specifies the default value of `indents` for newly created objects.

sysLevel

System Property

DESCRIPTION A system property that specifies the system working level.

NOTES You can get or set this property.

VALUES `Author` or `Reader`.

The default is the current working level.

For viewers with `alwaysReader` set to `true`, the viewer remains at `Reader` level regardless of the setting for `sysLevel`.

EXAMPLES

```
sysLevel = "Reader"
```

sysLineEndSize

System Property

DESCRIPTION A system property that specifies the default value of `lineEndSize` for all newly created objects.

sysLineEndStyle

System Property

DESCRIPTION A system property that specifies the default value of `lineEndStyle` for all newly created objects.

sysLineSpacing

System Property

DESCRIPTION A system property that specifies the default value of `spacing` for all newly created objects.

sysLineStyle

System Property

DESCRIPTION A system property that specifies the default value of `lineStyle` for all newly created objects.

sysLinkedDLLs

System Property

DESCRIPTION A system property that returns a list of DLLs that are currently linked to the ToolBook instance.

NOTES This property cannot be set.

Use this property to verify whether a particular DLL is linked to ToolBook, or which version of a DLL is linked when multiple versions of the DLL are available.

VALUES A list of path names that represent each DLL linked to the ToolBook instance.

sysLockScreen

System Property

DESCRIPTION A system property that specifies whether to allow screen updates during the execution of the current handler.

NOTES You can get or set this property.

When `sysLockScreen` is set to `true` in a handler, it is automatically reset to `false` when ToolBook exits the topmost calling handler.

If you set `sysLockScreen` to `true` in a script, you can make several changes in the book and then show all the changes at once, instead of waiting for ToolBook to update the screen after each change.

You can also use this property to force ToolBook to update the screen by first setting `sysLockScreen` to true, then to false.

Note that the `lockScreen` property of a viewer behaves similarly to this property, however the `lockScreen` property only affects one viewer whereas `sysLockScreen` affect all ToolBook viewers.

VALUES `True` or `false`.

The default is `false` (the screen updates after every change).

sysMediaBreakKey

System Property

DESCRIPTION A system property that specifies a key you can press to stop media played with the wait parameter.

NOTES You can get or set this property.

When you play a clip using the wait parameter, your system does not accept input while the clip plays. By setting a break key, you provide yourself (and your users) with an escape option.

VALUES Any valid key constant, except `keyLeftButton` and `keyRightButton`.

The default is `keyEscape` (`ANSI 27`).

To prevent users from stopping media that is playing, set `sysMediaBreakKey` to `0`.

sysMediaSuspend

System Property

DESCRIPTION A system property that specifies whether ToolBook displays an `Execution Suspended` message if a non critical error occurs while a media command is executing.

NOTES You can get or set this property.

The `sysMediaSuspend` property differs from `sysSuspend`; it responds only to multimedia-related errors.

Unlike `sysSuspend`, ToolBook does **not** automatically reset the value of `sysMediaSuspend` to `true` when the application returns to top -level.

VALUES `True` or `false`.

The default is `false`.

If `false`, the script continues to execute, and the error value is placed in `sysError` and `sysErrorNumber`.

If `true`, the script stops at the line causing an error, and ToolBook displays an `Execution Suspended` message.

sysMMEngineVersion

System Property

DESCRIPTION A system property that specifies the version of the ToolBook media engine.

NOTES This property cannot be set.

VALUES In ToolBook 8.5, the value is `8.50`.

sysNumberFormat

System Property

DESCRIPTION A system property that defines how numbers are formatted by the `format` command.

NOTES You can get or set this property.

The value of this property has no effect on the display of a number unless you also format the number with the `format` command.

Generally, you won't want to change the value of this property from its default of "?". This general precision format allows ToolBook to interpret `sysCurrency`, `sysThousand`, and `sysDecimal` characters embedded in a number.

When a number is formatted as `null` for use in a calculation, ToolBook strips `sysCurrency` and `sysThousand`, and converts `sysDecimal` to a period so the number is valid for calculation. ToolBook might not be able to strip characters other than the values of these properties, leading and trailing spaces, and percent symbols (%).

VALUES "?", `null`, or a string of placeholder symbols enclosed in quotation marks.

The default is "?".

For a table of the placeholder characters, see `format`.

sysOpenWindows

System Property

DESCRIPTION A system property that specifies a list of all open viewers in the current instance.

NOTES This property cannot be set.

VALUES A list of viewer references.

The ToolBook Main window (`viewer id 0`) is included in the list.

sysOpenMedia

System Property

DESCRIPTION A system property that specifies a list of all currently open media.

NOTES You cannot set this property.

VALUES A list of all media open on the system.

The list does not include media opened with the `callMCI()` function.

sysOptimizedSave

System Property

DESCRIPTION A system property that specifies if a book is optimized for CD-ROM when it is saved.

NOTES You can get or set this property.

A book can only be saved as optimized for CD-ROM at `Author` level.

Use this property if you are writing a script that saves a book as optimized for CD-ROM. You can also set the `buildCacheFile`, `cacheFileType`, and `sysOptimizedSave` properties using the `Save As` or `Save As EXE` dialog box.

VALUES `True` or `false`.

The default is `false` (the book is saved as normal).

sysOperatingSystem

System Property

DESCRIPTION A system property used to get the current operating system, graphical environment, and version number.

NOTES This property cannot be set.

VALUES

OS	VALUE
95	`Windows 95 4.0`
98	`Windows 98 4.10`
NT	`Windows NT 4.0`
2000	`Windows 2000 5.0`
ME	`Windows 98 4.90`
XP	`Windows XP 5.1`

sysOverrideParentPalette

System Property

DESCRIPTION A ToolBook window normally realizes its palette in the foreground, except when the active ToolBook window is a child window. When the active ToolBook window is a child window, it normally realizes its palette in the background. The `sysOverrideParentPalette` system property specifies whether a ToolBook application overrides this behavior for child windows.

NOTES This only applies for systems that are running using `256` colors.

For example, in ToolBook applications run using `Neuron`, the main ToolBook window is a child of the `Neuron` window, and its palette is normally realized in the background. This prevents palette flashing.

VALUES `True` or `false`.

The default is `false`.

When `sysOverrideParentPalette` is `false`, a ToolBook window that is a child window will realize its palette in the background.

When `sysOverrideParentPalette` is `true`, then all ToolBook windows will realize their palettes in the foreground.

When `sysOverrideParentPalette` is `false`, a ToolBook child window is displayed using the current system palette. It only adds its own colors if there are unused entries, otherwise it uses the closest matching colors that are already in the palette. When a ToolBook application is running in `Neuron`, for example, the current palette is determined by the browser. Using the default `sysOverrideParentPalette` value (`false`) prevents the palette flash that happens under certain conditions, but may degrade the appearance of the contents of the ToolBook page.

When `sysOverrideParentPalette` is `true`, the ToolBook application determines the optimum palette for displaying the page, and then replaces the current system palette (that was in effect before displaying the page). The ToolBook page is displayed at optimal quality, but palette flash may occur.

sysPageScroll

System Property

DESCRIPTION A system property t hat specifies the offset of the ToolBook Main window from the upper-left corner of a page.

NOTES You can get or set this property.

Setting the `sysPageScroll` is effectively the same as setting the `pageScroll of mainWindow`

VALUES A list of two non-negative numbers in `page units` representing the horizontal and vertical coordinates of the offset point, respectively, measured from the upper-left corner of the page. The coordinates can be set so that the Main window displays only the area inside a page.

sysPageUnitsPerPixel

System Property

DESCRIPTION A system property that specifies the number of `page units` per `pixel` horizontally and vertically.

NOTES This property cannot be set.

VALUES A list of two positive real numbers that represent the number of `page units` per `pixel` for the `x` and `y` axes.

For example, if ToolBook is running on a standard VGA monitor, the value of `sysPageUnitsPerPixel` is `15,15`.

sysPasswords

System Property

DESCRIPTION A system property that specifies a list of encrypted passwords for ToolBook to check before requesting a password from the user.

NOTES You can get or set this property.

If ToolBook is looking for an `Author`, `Open`, or `Save` password and the encrypted form of the password appears in the value of `sysPasswords`, ToolBook continues without asking for a password. Otherwise, ToolBook displays the Password dialog box.

Set this property to avoid having to enter a password every time you want to open or save a password-protected book, or switch to `Author` level in a password-protected book.

To get the encrypted value of a password, use the `ask password` command statement. The encrypted form of a string is placed in the special variable `IT` as a result of the `ask password` command.

VALUES Up to `10` encrypted passwords, each on a separate textline.

The default is `null`.

sysPattern

System Property

DESCRIPTION A system property that specifies the default value of `pattern` used for newly created objects.

sysPluginMode

System Property

DESCRIPTION A system property that specifies whether the current instance of ToolBook is running in plug-in mode (running in `Neuron`).

NOTES You cannot set this property.

VALUES `True` or `false`.

sysPolygonShape

System Property

DESCRIPTION A system property that specifies the number of sides for a `polygon` when it is drawn.

NOTES You can get or set this property.

VALUES An integer from `3` to `99`, representing the number of sides for a `polygon`.

The default is `4`.

sysReaderRightClick

System Property

DESCRIPTION A system property that specifies whether ToolBook displays an object's right-click menu when it is right-clicked at `Reader` level.

NOTES When no handler is present in the object hierarchy for a `rightButtonDown` message, and `sysReaderRightClick` is `true`, ToolBook displays the object's right-click menu.

Setting this property causes an error in Runtime ToolBook.

VALUES `True` or `false`.

The default is the value of the `startupReaderRightClick` property.

If no setting exists for `startupReaderRightClick`, the default for `sysReaderRightClick` is false.

If `false`, ToolBook's default response to a `rightButtonDown` message is to do nothing.

If `true`, ToolBook displays a right-click menu for the target object at `Reader` level when it receives the `rightButtonDown` message.

sysRGBfill

System Property

DESCRIPTION A system property that specifies the default value of `rgbFill` for newly created objects.

sysRGBstroke

System Property

DESCRIPTION A system property that specifies the default value of `rgbStroke` for newly created objects.

sysRuntime

System Property

DESCRIPTION A system property that specifies whether the engine being used to run the ToolBook file is the `Runtime Engine`.

NOTES This property cannot be set.

This property is useful for preventing users from using commands that cause errors in `Runtime ToolBook`.

VALUES `True` or `false`.

The value is `true` if the user is using `Runtime ToolBook`.

sysSecureMode

System Property

DESCRIPTION A system property that specifies whether the current instance of ToolBook is running in `secure mode`.

NOTES You can get, but not set, this property.

VALUES `True` or `false`.

This property will be `true` whenever the ToolBook application is running in ToolBook `Neuron` in `secure mode`.

See the `Neuron` documentation for more details on secure and non-secure modes.

sysSendToolBookMessages

System Property

DESCRIPTION Determines whether any new controls that are created send ToolBook messages or their OCX/VBX equivalents in response to user events.

NOTES You can get or set this property.

Changes to `sysSendToolBookMessages` have no effect on controls already created, but instead applies to OCX/VBX controls created after changing the value.

If an OCX/VBX message does not appear in the table below (such as `GotFocus` or `LostFocus`), the message is not translated into an equivalent ToolBook message; you must write handlers for the OCX/VBX message.

TOOLBOOK MESSAGE	OCX/VBX MESSAGE EQUIVALENT
`buttonClick`	`Click`
`buttonDoubleClick`	`DblClick`
`buttonDown`	`MouseDown <button> <isShift>,<xPos>,<yPos>`
`buttonUp`	`MouseUp <button> <isShift>,<xPos>,<yPos>`
`keyChar`	`KeyPress <key>`
`keyDown`	`KeyDown <keyCode>,<isShift>`
`keyUp`	`KeyUp <keyCode>,<isShift>`

The following ToolBook messages are always sent regardless of the `sysSendToolBookMessages` setting: `moved`, `destroy`, and `sized`.

ToolBook can translate OCX/VBX messages to standard ToolBook messages, only if the control sends the standard form (that is a message with the standard set of parameters) of the OCX/VBX message.

VALUES `True` or `false`; the default is `false`.

If `true`, OCX/VBX controls send ToolBook messages.

If `false`, they send the OCX/VBX messages listed in the table above.

EXAMPLES

```
sysSendToolBookMessages = true
sysSendToolBookMessages = false
```

sysShowMRUfiles

System Property

DESCRIPTION A system property that specifies whether the most recently used file list appears as a section of the File menu at `Author` level.

NOTES You can get or set this property.

VALUES `True` or `false`.

The default is `true`.

sysStrokeColor

System Property

DESCRIPTION A system property that specifies the default value of `strokeColor` for newly created objects.

sysSupportedMedia

System Property

DESCRIPTION A system property that specifies the supported media devices in the current system.

NOTES This property cannot be set.

VALUES A list of all supported media drivers and devices that are installed in the system.

While the list includes any media devices that are installed on the system, it does not necessarily ensure that every installed device will play properly.

EXAMPLES

```
request sysSupportedMedia
```

sysSuspend

System Property

DESCRIPTION A system property that specifies whether ToolBook displays the `Execution Suspended` message when an error occurs during execution of a script.

NOTES You can get or set this property.

When `sysSuspend` is set to `false`, script execution will continue even if an error occurs.

VALUES `True` or `false`. The default is `true`.

When `sysSuspend` is true, ToolBook suspends script execution in the event of an error and displays an `Execution Suspended` message.

When `sysSuspend` is `false`, ToolBook does not suspend script execution if an error occurs, but sets the `sysError` property to indicate the nature of the error.

You can define your own error handling by setting `sysSuspend` to `false`, then writing statements to check the value of `sysError` and respond appropriately.

sysSuspendMessages

System Property

DESCRIPTION A system property that specifies whether ToolBook sends `enter` and `leave` event and menu event messages automatically.

NOTES You can get or set this property.

VALUES `True` or `false`. The default is `false`.

When `sysSuspendMessages` is `false`, ToolBook sends `enter` and `leave` event and menu event messages as usual. When this property is `true`, ToolBook does not send system-generated messages, including all built-in event messages and all menu event messages.

ToolBook resets `sysSuspendMessages` to `false` when the topmost calling handler finishes executing.

Set this property to `true` to prevent ToolBook from sending messages such as `leavePage`, `leaveBook`, `enterBook`, and `enterPage` when navigating to other pages or books.

sysSystemVariables

System Property

DESCRIPTION A system property that returns a list of all system variables that are currently defined in the ToolBook instance.

NOTES This property cannot be set.

VALUES An alphabetically sorted list of system variable names in the current ToolBook instance.

sysTabSpacing

System Property

DESCRIPTION A system property that specifies the default value of `tabSpacing` for newly created objects.

sysTabType

System Property

DESCRIPTION A system property that specifies the default value of `tabType` for newly created objects.

system

Variable Command

SYNTAX
```
system [<type>] <variable list>
local [<type>] [fixed|dynamic] <variable name>[<size>][<size>]...
```

DESCRIPTION Makes a variable name known and its contents available to any script in the current instance of ToolBook, including the scripts of any viewers shown by that instance. Changing the value of a `system` variable in any script changes its value in every script that uses the variable.

You must use the `system` command in each handler in which a `system` variable is used, or ToolBook treats the variable as a `local` variable that is valid only within that handler.

ToolBook supports the declaration of data types for `local` and `system` variables, and optional data types for `local` and `system` arrays. Use of data types produces more efficient code and faster script execution. Use of typed arrays improves array performance.

ToolBook initializes the value of a non-typed `system` variable to `null` when it is declared. ToolBook initializes the value of a `system` variable according to its data type.

DATA TYPE	VALUE
DWORD, INT, LONG, REAL, WORD	`0`
LOGICAL	`false`
POINT	`0,0`
COLOR	`0,0,0`
STACK, STRING	`null`
all other types	`null`

A `system` variable is available to any script in the ToolBook *instance* in which you have defined it. However, a `system` variable defined in one instance of a book is not available to another instance of the same book. A handler cannot contain both a `system` variable and a `local` variable with the same name.

The following operations function most efficiently when you use the suggested data type:

USE	DATA TYPE
Numeric operations	`REAL`
Text operations	`STRING`
Logical operations	`LOGICAL`
Step loops	`LONG`

PARAMETERS

PARAMETER	DESCRIPTION
`<variable list>`	One or more variable names.
`<type>`	The name of the data type.
`fixed | dynamic`	Declares a local array as either fixed or dynamic. A fixed array cannot be reset or expanded.
`<size>`	The number of elements in a dimension of a local array. Note that ToolBook is limited to a 16 dimensional array, as in: `var[][][][][][][][][][][][][][][]`

EXAMPLES

```
-- Place these handlers in a book script to save a field's
-- text in a system variable when focus enters the field,
-- then check to see if text changed when focus leaves the field
to handle enterField
    system fieldText
    put the text of the target into fieldText
    forward
end

to handle leaveField
    system fieldText
    if fieldText is not the text of the target
        send update
    end
    forward
end

-- Declares a typed system variable
system LOGICAL vVariable

-- Declares a dynamic system array variable
system LOGICAL dynamic vArray[][]
```

sysTime

System Property

DESCRIPTION A system property used to get the current system time, which is set by the computer.

NOTES This property cannot be set.

VALUES The current system time in the format specified by `sysTimeFormat`.

sysTimeFormat

System Property

DESCRIPTION A system property that defines the format of `sysTime` and how time is formatted by the `format` command.

NOTES You can get or set this property.

The value of this property only affects the value of `sysTime`, unless the time is also formatted with the `format` command.

To use a time in an arithmetic calculation, format the time as "`seconds`".

VALUES A string of placeholder characters enclosed in quotation marks.

The default is based on the values of `sysTime` and `sysTimeChar`.

For a table of the placeholder characters, see `format`.

sysTool

System Property

DESCRIPTION A system property that specifies the currently selected tool on the `tool palette` at `Author` level.

NOTES You can set this property to change the currently selected tool or get the property to return the currently selected tool.

The value of `sysTool` at `Reader` level is "`Reader`".

VALUES

Getting this property returns one of the following values:		
angledLine	extensionEdit	radioButton
arc	field	radioButton3D
borderlessField	irregularPolygon	recordfield
button	labelButton	rectangle
checkBox	line	roundedRectangle
checkBox3D	magnify	select
comboBox	ole	singleSelectListBox
curve	pie	stage
ellipse	polygon	

Getting this property at `Reader` level returns the value of `Reader`.

When you set this system property to change the tool, the change is reflected on the tool palette.

sysToolTips

System Property

DESCRIPTION An Author level setting that controls the display of `tooltips` when the mouse cursor is over a button on any of the following: tool palette, toolbar, script editor toolbar.

NOTES This property is not available in the `runtime` version.

This property is not associated with the `tooltips` setting you can configure for object in your book.

VALUES `True` or `false`.

When true `tooltips` are displayed.

When false `tooltips` will not be displayed.

sysToolBookDirectory

System Property

DESCRIPTION A system property that specifies the path of the directory in which a ToolBook instance is currently running.

NOTES This property cannot be set.

Typically the ToolBook Engine will be found in the `tbsystem` directory.

VALUES The complete path of the ToolBook Engine.

The ToolBook Engine will be either `INSTRUCTOR.EXE` or `TB89RUN.EXE` depending on if you are using the Authoring Engine or the Runtime Engine.

The path ends with a single backslash (\) character.

sysTransparent

System Property

DESCRIPTION A system property that specifies the default value of `transparent` for newly created objects.

sysUnits

System Property

DESCRIPTION A system property that specifies the unit of measure used for rulers, grid spacing, page size, tabs, and indents.

NOTES You can get or set this property.

VALUES `English` or `metric`.

The default is based on the `sysIMeasure` value.

If `english`, the measurements are in inches.

If `metric`, the measurements are in centimeters.

sysUseWindowsColors

System Property

DESCRIPTION A system property that specifies the default value of `useWindowsColors` for newly created objects.

sysVersion

System Property

DESCRIPTION A system property that specifies the version of ToolBook or ToolBook that is currently running.

NOTES This property cannot be set.

VALUES The string value `8.5 United States`

sysWindowHandle

System Property

DESCRIPTION A system property that specifies the `16-bit` window handle of the ToolBook Main window.

NOTES This property cannot be set.

Getting the value for `sysWindowHandle` is the equivalent of getting the value of the `windowHandle of mainWindow`.

VALUES The `16-bit` window handle of the ToolBook Main window.

sysWindowHandle32

System Property

DESCRIPTION A system property that specifies the `32-bit` window handle of the ToolBook Main window.

NOTES This property cannot be set.

Getting the value for `sysWindowHandle32` is the equivalent of getting the value of the `windowHandle32 of mainWindow`.

VALUES The `32-bit` window handle of the ToolBook Main window.

tab

Spacing Constants

DESCRIPTION Represents the tab ANSI character. This is equivalent to ANSI 9.

EXAMPLES

```
-- remove leading tabs from a text string
while first character of var = tab
    clear first char of var
end
```

```
a = b & tab & c
```

tabSpacing

Property

DESCRIPTION A field or recordfield property that specifies the tab stop settings.

NOTES You can get or set this property.

Notice that you can only set one tabSpacing per field, rather than several per paragraph (textline) as you can in Microsoft Word. This is an inherent limitation of ToolBook fields.

VALUES A positive whole number in page units specifying the interval for tab stops in a field or recordfield.

The default is the value of sysTabSpacing.

tabType

Property

DESCRIPTION A field or recordfield property that specifies whether left or decimal tabs are used in a field or recordfield.

NOTES You can get or set this property.

Notice that you can only set one type per field, rather than one type per paragraph (textline) as you can in Microsoft Word. This is an inherent limitation of ToolBook fields.

VALUES Left or decimal.

The default is the value of sysTabType.

tan()

Trigonometric Values

SYNTAX

```
tan(<angle>)
```

DESCRIPTION Returns the tangent of an angle measured in radians.

The formula for converting degrees to radians is radians = degrees * (pi/180).

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<angle>	An expression that yields a number.

EXAMPLES

```
x = tan(0.785)       -- equals 0.99920
x = tan(45*pi/180)   -- equals 1
```

tanh()

Trigonometric Values

SYNTAX `tanh(<angle>)`

DESCRIPTION Returns the hyperbolic tangent of an angle measured in radians.

The formula for converting degrees to radians is `radians = degrees * (pi/180)`.

PARAMETERS

PARAMETER	DESCRIPTION
`<angle>`	An expression that yields a number.

EXAMPLES

```
x = tanh(-2)   -- equals -0.96403
x = tanh(0)    -- equals 0
x = tanh(0.5)  -- equals 0.462117
```

target

System Property

DESCRIPTION A system property that specifies which object first received the current message.

NOTES This property cannot be set.

Use `target` in a group script handler to refer to the object in the group that first receives the message.

Because target is a system property, you can get its value more quickly than getting the return value from `objectFromPoint()`.

Sending a message to another object with the `send` command changes the value of `target`.

Forwarding a message with the `forward` command does not change the value of `target`.

VALUES The `uniqueName` of the object that received the current message.

targetWindow

System Property

DESCRIPTION A system property that specifies the viewer in which ToolBook executes commands and searches for objects.

NOTES You cannot set this property.

VALUES A viewer reference.

ToolBook executes commands on the objects, page, and background displayed in the viewer that is the target window.

To execute a command within the context of a particular viewer, make sure the viewer is open and it is the target window.

To specify the value of `targetWindow`, use the `in <viewer reference>` control structure with a viewer reference for the target window, or use the `in <viewer reference>` parameter with the `send` command.

DESCRIPTION What follows is a list of error return codes that might be passed back to OpenScript as a result of executing one of the functions encapsulated in `TBFILE32.DLL`.

RETURNS

CODE	DESCRIPTION
1	true or success
0	false
-1	failed
-2	user cancelled
-3	undetermined error
-4	internal error
-5	file I/O error
-6	can't open source
-7	can't open destination
-8	file not found
-9	invalid path
-10	invalid drive
-11	invalid drive letter
-12	access denied
-13	tag does not exist
-14	illegal string
-15	invalid attribute
-16	invalid sort order
-17	invalid parameter
-18	invalid name
-19	Path or file name too long
-20	current directory specified
-21	different drives for source and destination
-22	in path but not current directory
-23	network problem
-24	can't link to 32 bit DLL
-25	buffer too small
-26	too much data for buffer
-27	out of memory
-28	too many files open
-29	disk full
-30	security problem
-31	cannot make file or directory
-32	directory not empty
-33	directory or file already exists
-34	file cannot be copied onto itself
-35	file spec too long for DOS
-36	invalid file spec component
-37	invalid character in filespec
-38	drive not ready

tenth

Ordinal Numbers

DESCRIPTION A value used to indicate the tenth item in a sequence or list. Also used to indicate relative position of pages or backgrounds.

EXAMPLES

```
-- The following paired examples show how you can use ordinal
-- references to make OpenScript easier to read. As you can see it is
-- possible to write your scripts with or without ordinal references.
go to the tenth page of this book
go to page 10 of this book

if tenth character of str = "X"
if character 10 of str = "X"

tenth word of text of field "lesson" = "Hello"
word 10 of text of field "lesson" = "Hello"

put "Cancelled:" into tenth character of name of button id 16
put "Cancelled:" into character 10 of name of button id 16
```

text

Property

DESCRIPTION A combobox, field, hotword, or recordfield property used to set, get, or manipulate the object's text.

NOTES You can get or set this property.

The `text` property of these objects contain the unformatted raw characters that make up the text string. The `richText` property on the other hand contains `RTF` formatted text, which not only holds the raw text but also the formatting too.

For `comboboxes`, the value of the text property refers to the text in the edit box, not the text in the dropdown box.

VALUES A string consisting of the text in a specified combobox, field, recordfield, or hotword.

The maximum number of characters that this value can hold is 32,000.

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES

```
if text of field "printerInfo" <> null
    put "hello" into text of combobox "listing"
end

text of recordfield "data" of page "b" = text of field "r3"

request characters 4 to 16 of text of field "element"
```

textAlignment

Property

DESCRIPTION A field or recordfield property that specifies the alignment of text in a field or recordfield.

NOTES You can get or set this property.

Notice that you can only set one alignment type per field, rather than one type per paragraph (textline) as you can in Microsoft Word. This is an inherent limitation of ToolBook fields.

VALUES `Left`, `right`, `justify`, or `center`.

The default is the value of `sysAlignment`.

textFromPoint()

String Functions

SYNTAX

```
textFromPoint(<location>)[of <field reference>]
```

DESCRIPTION

Determines which character in a field/recordfield is currently at a specific `x,y` location.

RETURNS

Returns a list of two numbers indicating the textline number and character position in a field or recordfield at a specified location.

Note that a textline in a field or recordfield is terminated by a `CRLF`. Don't confuse this with word wrapping.

The first number is the number of textlines from the beginning of the text to the textline containing the character at the specified location.

The second number is the number of characters from the beginning of the textline containing the specified location to the character at the specified location.

For example, if the user clicks the second character in the third textline of a field, the function returns `3,2`.

ToolBook always evaluates this function in the context of a specific field. If no character is at that location, this function returns `-1,-1`.

PARAMETERS

PARAMETER	DESCRIPTION
`<location>`	A list of two numbers that define a location in `page units`.
`<field reference>`	Optional Parameter. A reference to a field or recordfield.

EXAMPLES

```
-- Put this handler in a field's script to select the paragraph
-- (textline)a user clicks; the field must be activated for this
-- script to work
to handle buttonClick loc
    get item 1 of textFromPoint(loc)
    if IT > 0
        select textline IT of my text
    end
end

-- Put this handler in a page's script to cause ToolBook to delete
-- any character the user clicks in any activated field on that page
to handle buttonClick loc
    --Checks if the user clicked a field
    if object of target = field
        get textFromPoint(loc) of target
        if item 1 of IT > 0
            clear character (item 2 of IT) of textline (item 1 of IT) \
              of text of target
        end
    end
end
```

textline, textlines

String Operators

SYNTAX

```
textline <number> of <stringListRef>
textlines <number> to <number> of <stringListRef>
```

DESCRIPTION

Used to refer to specific textlines in a text string. A `textline` is defined as a sequence of characters surrounded by a carriage return (`CRLF`).

The first and last textline in a string are the exceptions to this rule, as the first `textline` will unlikely have a `CRLF` preceding it, as is the last `textline` in a string unlikely to have a `CRLF` following it.

NOTES

Various terms can be used with `textline` including `first`, `last`, `middle`.

Only a `CRLF` is used as an textline delimiter. It is not possible to tell ToolBook to use a different delimiter.

PARAMETERS

PARAMETER	DESCRIPTION
<number>	An expression that results in a number.
<stringListRef>	An expression that results in a string value.

EXAMPLES

```
-- trim off empty textlines
if last textline of str = null
    clear last textline of str
end
```

```
x = textlines 1 to 5 of text of field "list"
```

```
-- Position all objects on the current page to the 0,0 position
objs = listToTextline(objects of this page)
step k from 1 to textlineCount(objs)
    curObj = textline k of objs
    position of curObj = 0,0
end
```

textlineContains()

TBDLG.DLL String Functions

SYNTAX

```
textlineContains(<string1>,<string2>)
```

DESCRIPTION

Searches <string2> to see if <string1> can be found.

RETURNS

If a match is found, the textline number of the first match is returned. If no match occurs then the value of 0 is returned.

NOTES

As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    INT textlineContains(STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
<string1>	An expression that results in a string value.
<string2>	An expression that results in a string value.

EXAMPLES

```
to handle buttonClick
    -- Part number TLx100-11 is on textline 16
    x = text of field "part numbers"
    loc = textlineContains("x100",x)
    if loc > 0
        -- Match Found on line loc
        -- return full line text
        request "First match found:" && textline loc of x
    end
end
```

textlineCount()

SYNTAX `textlineCount(<expression>)`

DESCRIPTION Counts the number of textlines in an expression.

RETURNS Returns the number of textlines in a string. A `textline` is defined as a sequence of characters surrounded by the textline terminator `CRLF`.

The first and last textline in a string are the exceptions to this rule, as obviously the first textline will unlikely have a `CRLF` preceding it, as is the last textline in a string unlikely to have a `CRLF` following it.

NOTES You can also count characters with `charCount()` and words with `wordCount()`.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that results in a string value.

EXAMPLES

```
-- Check to see if Poem is long enough
to handle buttonClick
    ttl = textlineCount(text of field "poem")
    if ttl < 25
        request "You must have at least 25 lines in your poem."
    end
end
```

textlineOffset()

SYNTAX `textLineOffset(<line>, <text>)`

DESCRIPTION Locates the first occurrence of a textline in a string.

RETURNS Returns the textline position in which `<line>` was found in `<text>`. If `<line>` cannot be found within `<text>` then `0` is returned.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    INT textlineOffset(STRING,STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<line>`	An expression that results in a string value.
`<text>`	An expression that results in a string value.

EXAMPLES

```
to handle buttonClick
    lst = "apple" & CRLF & "banana" & CRLF & "orange"
    ask "What are you looking for?"
    answ = IT
    pos = itemOffset(answ,lst)
    if pos > 0
        request "Yes, that item is in textline position" && pos
    else
        request "Sorry, didn't find a match"
    end
end
```

textlineToList()

TBDLG.DLL String Functions

SYNTAX `textlineToList(<textlines>)`

DESCRIPTION This function converts your string of textlines into a string of items.

RETURNS This conversion occurs by the simple replacement of all `CRLF` characters in a string with commas.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbdlg.dll"
    STRING textlineToList(STRING)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<textlines>`	An expression that results in a string value.

EXAMPLES

```
to handle buttonClick
    x = text of field "parts list"
    request textlineToList(x)
end
```

textOffset

Property

DESCRIPTION A hotword property that specifies the character position of the first character of a hotword's text in a field or recordfield.

NOTES This is a read only property and cannot be set.

VALUES A positive whole number that specifies the character position of the hotword's first character.

EXAMPLES

```
hw = hotword "URL Linker" of field "list"
request chars 1 to (textOffset of hw - 1) of text of field "list"
```

textOverflow

Property

DESCRIPTION A button, combobox, field, or recordfield property that specifies the number of characters completely or partially clipped.

NOTES This is a read only property and cannot be set.

This property is useful for determining whether all of the text fits in the display area of an object.

VALUES A non-negative whole number that equals the number of characters of a button's, combobox's, field's, or recordfield's text that are partially or completely clipped.

For buttons, comboboxes, and single line fields and recordfields, this property reports the number of characters clipped at the right edge.

For multi line fields and recordfields, the property reports the number of characters clipped at the bottom edge.

For button labels, the value represents the number of characters clipped at the button's right or bottom edge.

For any style of field or recordfield, the value of `textOverflow` represents the number of characters clipped at the bottom of the object.

For single line fields, the value represents the number of characters clipped at the right edge of the field.

textRightOverflow

Property

DESCRIPTION A field or recordfield property that specifies the number of characters completely or partially clipped for each visible line of text.

NOTES This is a read only property and cannot be set.

VALUES A comma separated list of non-negative whole numbers representing the number of partially or completely clipped characters for each displayed line in the field. The `itemCount` of the list is equal to the number of textlines displayed.

The first number in the list corresponds to the first visible line of text.

Use this property to determine whether all of the text fits in the display area of a `singleSelect`, `multiSelect`, or `noWrap` field type.

For a `wordWrap` field type, the numbers in the list are always `0`.

textScrolled

Notification Message

SYNTAX `textScrolled <newPos>`

DESCRIPTION Sent to a field when the user clicks the field's `scroll bar`.

The field's `borderStyle` property must be set to `scrolling` for the `textScrolled` message to occur.

NOTE Be aware that no message is sent if a user scrolls a field by using arrow keyboard keys, or by clicking and dragging up or down in a text field.

As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

PARAMETERS

PARAMETER	DESCRIPTION
`<newPos>`	A non-negative integer that is the same value as the field's `scroll` property.

EXAMPLES

```
-- Place these handlers in a page script or higher
-- to synchronize scrolling between two list box fields
to handle textScrolled
    forward
    send synch
end

to handle synch
    if focus is field "dayData" or focus is null
        scroll of field "times" = scroll of recordfield "dayData"
    else
        scroll of field "dayData" = scroll of field "times"
    end
end
```

textToPrinter()

TBWIN.DLL Printing Function

SYNTAX `textToPrinter(<string>,<options>,<window handle>,<bShowDlg>)`

DESCRIPTION Prints a text string to the current default printer.

This function automatically wraps the text to fit on the printed page, with fixed (8-space) tabs. Only one style of formatting is available for each document.

RETURNS If successful, the function returns `true`. Otherwise, it returns `false`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT textToPrinter(STRING,STRING,WORD,INT)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<string>`	The text string to print.
`<options>`	A list of textlines (`CRLF`-separated) that specify the following options: `Document Name` — A name for the document to be printed. `Font Face` — A valid font name. `Font Style list` — Bold, italic, underline, or strikeout. `Font Size` — A valid font size. `Margins in inches` — A string that contains a list of four numbers that correspond to the `<left>`, `<top>`, `<right>`, and `<bottom>` margins.
`<window handle>`	The window handle of the parent window of the Cancel dialog box.
`<bShowDlg>`	`1` Shows the Cancel dialog box, `0` Does not show the Cancel dialog box.

EXAMPLES

```
to handle buttonClick
    txt = text of field "sample"
    docName = "Test"
    fontF = "Times New Roman"
    fontS = "Bold"
    fontSz = 10
    marg = "1.5,1,1.5.1"
    opts = docName & CRLF & fontF & CRLF & fontS & CRLF & fontSz & CRLF & marg
    hndl = windowHandle of mainWindow
    showDlg = 1
    get textToPrinter(txt,opts,hndl,showDlg)
end
```

textUnderflow

Property

DESCRIPTION A button, combobox, field, or recordfield property that specifies the number of characters completely or partially clipped at the top or left edge of the object.

NOTES This is a read only property and cannot be set. This property is useful for determining whether all of the text fits in the display area of an object.

VALUES A non-negative whole number that equals the number of characters of a button's caption or combobox's, field's, or recordfield's text that are partially or completely clipped.

If no characters are clipped, the value is `0`.

the

Articles and Prepositions

DESCRIPTION Used to make OpenScript more readable. This terms is optional in every context in which it can be used.

EXAMPLES

```
go to the last page of this background

put the text of field "apple" into the name of this page
```

third

Ordinal Numbers

DESCRIPTION A value used to indicate the third item in a sequence or list. Also used to indicate relative position of pages or backgrounds.

EXAMPLES

```
-- The following paired examples show how you can use ordinal
-- references to make OpenScript easier to read. As you can see it is
-- possible to write your scripts with or without ordinal references.
go to the third page of this book
go to page 3 of this book

if third character of str = "X"
if character 3 of str = "X"

third word of text of field "lesson" = "Hello"
word 3 of text of field "lesson" = "Hello"

put "Cancelled:" into third character of name of button id 16
put "Cancelled:" into character 3 of name of button id 16
```

this

Articles and Prepositions

DESCRIPTION An term used in conjunction with `background`, `book`, `page`, and `window` to indicate the current object in the context of the `Target Window`.

EXAMPLES

```
go to the last page of this background

if the name of this page contains "no-Print"
    request "Sorry, you are not permitted to print this page"
end

if chars 1 to 3 of name of this book <> "C:\"
    request "This book must be opened from the C Drive"
end

to handle buttonClick
    close this window
end
```

tile

Property

DESCRIPTION A property of the `tool bar` or `tool palette`, or a viewer that specifies the position of a palette or child window relative to its parent window's client window.

The Main window is the parent window of built-in ToolBook palettes.

NOTES The `tile` property can be set only for viewers with type set to `child`. The `tile` property is ignored for viewers that are popup windows and for the ToolBook Main window.

VALUES None, top, bottom, left, or right. The default is none.

For palettes: if none, the tool bar or tool palette is positioned according to its last position on the screen. When tile is set to any value except none, the palette clings to the specified side of the Main window. ToolBook tiles palettes according to their layer order.

For viewers: if tile is set to none when a viewer first opens, it is positioned and resized according to the settings for its defaultPosition and defaultClientSize properties.

If top or bottom, the viewer is resized to fit horizontally within the bounds of the client window of its parent window, and ToolBook moves the viewer to the specified edge. ToolBook increases the size of the parent window to accommodate the tiled viewer.

If left or right, the viewer is sized to fit vertically within the bounds of the client window of its parent window, and ToolBook moves the viewer to the specified edge.

EXAMPLES

```
tile of viewer "help" = "bottom"
```

tileOrder

Property

DESCRIPTION A property of the tool bar or tool palette or a property of a viewer that specifies tiling order for a child window or palette that are configured to tile.

NOTES The tileOrder property applies to palettes and child windows with the tile property set to any value but none.

For viewers, you can get or set the tileOrder property only for child windows (viewers with defaultType set to child). The tileOrder property cannot be set for viewers that are popup windows.

VALUES A non-negative integer between 0 and 32767.

When a viewer that is a child window is opened, and its tile property is set to any value other than null, ToolBook assigns it to the next lowest layer from the top, starting at 1.

To move a viewer to the top of the layer order, set its tileOrder property to 0.

EXAMPLES

```
tileOrder of viewer "help" = 0
```

tileWrap

Property

DESCRIPTION A property of the tool bar or tool palette that specifies if the tool bar or tool palette wraps when it is tiled.

VALUES True or false.

If true, the tool bar or tool palette wraps to the left border of the viewer in which it is tiled.

If false, and if the tool bar or tool palette is wider than the current width of the viewer in which it is tiled, the tool bar or tool palette is truncated at the right edge of the viewer.

timerCapability()

Timer Function

SYNTAX

```
timerCapability()
```

DESCRIPTION Requests information about the capability of the timer device.

RETURNS Returns a list of two items in milliseconds.

The first item is the minimum resolution of the timer device; the second item is its maximum period.

PARAMETERS No parameters.

EXAMPLES

```
minRes = item 1 of timerCapability()
```

timerNotify

Timer Message

SYNTAX `timerNotify <timer ID>`

DESCRIPTION Sent to the object specified by the `<notify object>` parameter of the `timerStart()` function.

The `timerNotify` message is sent at the interval specified for the `timerStart()` function.

PARAMETERS

PARAMETER	DESCRIPTION
`<timer ID>`	A positive number that identifies the timer.

EXAMPLES

```
to handle timerNotify timerID
    system statusTimer
    if timerID = statusTimer
        text of field "CD position" = sysTime
    end
end
```

timerStart()

Timer Function

SYNTAX `timerStart(single|periodic,<delay>,<resolution>,<notify object>)`

DESCRIPTION Provides OpenScript access to the timer services supported by Microsoft Windows

RETURNS Returns the handle/ID to the specific timer. The value of this handle/ID is not important in itself, but is useful in cases where you may need to Stop that specific timer or determine which of various running timers has just expired.

If an error occurs, returns `null` and sets the value of `sysErrorNumber`.

PARAMETERS

PARAMETER	DESCRIPTION
`single | periodic`	Set the timer type to `single` for a one-time notification when the timer expires, or to `periodic` for periodic notification each time the timer expires. If a timer type is `periodic`, you must explicitly stop the timer with `timerStop()`.
`<delay>`	Sets the timer period to the specified number of `milliseconds`, after which the timer expires.
`<resolution>`	Sets the timer `resolution` to the specified number of `milliseconds`, which determines the accuracy with which the timer attempts to execute the timer callback function.
`<notify object>`	A reference to the object to be notified with the `timerNotify` message when the requested operation is completed. To prevent notification, do not include this parameter.

EXAMPLES

```
to handle buttonClick
    if timerID of self = null
        timerID of self = timerStart("periodic",5000,1000,self)
    end
    --Further statements here
end
```

timerStop()

Timer Function

SYNTAX `timerStop(<timerID>)`

DESCRIPTION Stops a timer that had been started with `timerStart()`.

RETURNS Returns `0` if the function is successful.

If an error occurs, returns `null` and sets `sysErrorNumber`.

PARAMETERS

PARAMETER	DESCRIPTION
`<timer ID>`	Must be the same value returned by the `timerStart()` function to stop the timer. The value can also be `0` to indicate that all timers should be stopped.

EXAMPLES

```
to handle rightButtonUp
    get timerStop(timerID of self)
    clear timerID of self
end
```

title

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of a book that specifies the Book Title entered in the Properties for Book dialog box.

NOTES You can get but not set this property in OpenScript as well as in the Actions Editor.

In OpenScript, this can also be achieved by manipulating the `info_Title` book user property.

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

EXAMPLES

```
ttl = title of this book
```

to

Articles and Prepositions

DESCRIPTION Used with the following OpenScript terms for various reasons: add menu, add menuItem, draw, break, forward, fxDissolve, fxWipe, fxZoom, go, move, readFile, save changes, select all, send, set, sort, step, transition, and writeFile.

EXAMPLES

```
if errorStatus <> null
    break to system
end

to handle buttonClick
    increment clickCount of this book
    forward
end

set text of field "fileList" to null

writeFile "LastEntry" to "C:\logfile.txt"
```

to handle

SYNTAX

```
to handle <message name> [<parameters>]
    <statements>
end [<message name>]
```

DESCRIPTION

Defines what will happen when an object receives a particular message.

NOTES

Feel free to create your own user defined messages - sending them to objects and receiving them with the `to handle` structure.

PARAMETERS

PARAMETER	DESCRIPTION
`<message name>`	The name of any `built-in` or `user-defined` message. All message names must begin with a letter and cannot include non-alphanumeric punctuation characters, except the underscore character.
`<parameters>`	A list of any parameters to be passed with the command or message.
`<statements>`	One or more OpenScript statements to be executed when the object receives the specified message name.

EXAMPLES

```
to handle buttonClick
    send displayMyPersonalPopupMessage "Hello World!", 2
end

to handle displayMyPersonalPopupMessage msg, delay
    pause delay seconds
    request msg
end
```

to get

SYNTAX

```
to get <name> [<parameter>[,<parameter>] ... ]
    <statements>
    return <value>
end [<name>]
```

DESCRIPTION

Defines a procedure for getting the value of a user-defined function or user property.

You must use `to get` handlers to create user-defined functions and to control what happens when a statement gets or refers to the value of a user property. The `return` statement acts as a `break` for the function. No statements following the `return` statement are executed.

The `forward` statement will pass the `to get` message up the object hierarchy so that it can be handled by other `to get` handlers. If the message reaches the ToolBook system, the value of the user property with the specified name will be returned. When the `forward` command is used, the return value from the forward statement is placed in the special variable `IT`.

NOTES

Feel free to create your own user defined messages - sending them to objects and receiving them with the `to handle` structure.

PARAMETERS

PARAMETER	DESCRIPTION
`<name>`	A `unique name` for the function or property being defined.
`<parameters>`	A list of one or more parameters that can be passed with the function or property.
`<statements>`	One or more OpenScript statements that define the function or property. These statements must include a `return` statement with the value to be returned.

EXAMPLES

```
to handle buttonClick
    request "What is your full name?"
    request "So, you say you name is: " & quoteIT(IT)
end

to get quoteIT str
    val = QUOTE & str & QUOTE
    return val
end
```

to set

Handler Structure

SYNTAX

```
to set <name> [<parameters>] to <value>
    <statements>
end [<name>]
```

DESCRIPTION

Defines a procedure for setting the value of a `user property`.

NOTES

This is used for user properties. It is not possible to define a process for setting the value of a Standard object property.

The `forward` statement will pass the `to set` message up the object hierarchy so that it can be handled by other to set handlers.

If the message reaches the ToolBook system, a `user property` of the specified name will be created for the target object.

Hint: Typical use of this handler does not include the use of the `<parameters>` optional setting, but it is a powerful ability. Refer to the second example below for a demonstration in using it.

PARAMETERS

PARAMETER	DESCRIPTION
`<name>`	A unique name for the user property.
`<parameters>`	A list of one or more parameters that can be passed with the property.
`<value>`	The value being set.
`<statements>`	One or more statements to be executed. To avoid creating an `infinite loop`, avoid referring to `<name>` in the body of the handler.

EXAMPLES

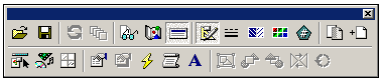
```
-- Sample showing typical use of a to set handler
to handle buttonClick
    fullName of self = "John Doe"
end

to set fullName to str
    text of self = str
    if str <> null
        fillColor of self = yellow
    else
        fillColor of self = white
    end
end

-- Sample showing the seldom utilized use of the <parameters> setting
to handle buttonClick
    fullName("John","Doe") of self = "CapitalizeIt"
end

to set fullName fName, lName to val
    str = fName && lName
    if val = "CapitalizeIt"
        request upperCase(str)
    else
        request str
    end
end
```

toolBar

System Object

DESCRIPTION The object type name for the `tool bar` in the ToolBook Main window.

NOTES You can show and hide the tool bar using the `show` and `hide` commands, or by setting its `visible` property.



By default, the tool bar appears in the top portion of the ToolBook Main window. You can also detach the tool bar and resize it as a palette, or attach the tool bar to a different side of the Main window.

To set the tool bar's default position in the Main window, set the `tile` property of `toolBar` to `none`, `top`, `bottom`, `left`, or `right`.

To specify whether the tool bar displays `captions`, or `graphics`, or both `captions` and `graphics` simultaneously, set the `style` property of `toolBar` to either `captions` or `graphics`, or to `graphics, captions`.

The tool bar is visible at `Author` level by default and cannot be shown at `Reader` level.

An error results if you attempt to show the tool bar in Runtime ToolBook.

PROPERTY	VALUES
`bounds`	List of four whole numbers in `pixels`.
`position`	List of two whole numbers in `pixels`.
`style`	`Captions`, or `graphics`, or a list of both: `graphics, captions`.
`tile`	`None`, `top`, `bottom`, `left`, `right`; default is `top`.
`tileOrder`	A whole number between `0` and `32767`.
`tileWrap`	`True`, `false`; default is true
`vertices`	List of four whole numbers in `pixels`.
`visible`	`True` or `false`.

toolPalette

System Object

DESCRIPTION The object type name for the `tool palette`, where the tools for creating ToolBook objects reside.

NOTES When the tool palette is displayed, you can use mouse shortcuts to display other palettes, magnify or reduce the view, and select objects.

Use the `hide`, `show`, and `move` commands to control the `visibility` or `position` of the tool palette. By default, the tool palette appears as a floating palette in the ToolBook Main window.

You can also resize the tool palette, or attach the tool palette to any side of the Main window.

To set the tool palette's default p osition in the Main window, set the tile property of toolPalette to `none`, `top`, `bottom`, `left`, or `right`.

To specify whether the tool palette displays `captions`, or `graphics`, or both `captions` and `graphics` simultaneously, set the `style` property of `toolBar` to either `captions` or `graphics`, or to `graphics, captions`.

You cannot show the tool palette in Runtime ToolBook.

PROPERTY	VALUES
bounds	List of four whole numbers in pixels.
position	List of two whole numbers in pixels.
style	Captions, or graphics, or graphics,captions.
tile	None, top, bottom, left, right; default is none.
tileOrder	An integer between 0 and 32767.
tileWrap	True or false.
vertices	List of four whole numbers in pixels.
visible	True or false.

top

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of almost all ToolBook object types that specifies the current top position of that object.

NOTES You can get or set this property in OpenScript as well as in the Actions Editor.

In OpenScript, this can also be achieved with: `item 2 of position of <target>`

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

VALUES A whole numbers in `page units`.

EXAMPLES

```
top of button "foo" = 500
```

```
x = top of button "foo"
```

translateWindowMessage

Control Structure

SYNTAX

```
translateWindowMessage [for <winHandle>]
    on <winMsg> get <tbkMsg> [of <tbkObj>] return <dllType>
    --Further statements here
end [translateWindowMessage]
```

```
translateWindowMessage [for <winHandle>]
    --Further statements here
    on <winMsg> send <tbkMsg> [to <tbkObj>]
end [translateWindowMessage]
```

DESCRIPTION A control structure that establishes Windows-to-OpenScript message translation and specifies whether the OpenScript message is processed instead of the window's default processing of a message.

NOTES Use the `forward to system` statement in the handler called by the `on` statement in the `translateWindowMessage` control structure to forward a Windows message to the ToolBook system. ToolBook calls the default Windows procedure and puts the returned value into `IT`.

When you forward a translated message to the ToolBook system, ToolBook calls the default Windows procedure for the message at the time the message is forwarded. If a translated message is not forwarded to the system, ToolBook does not call the default Windows procedure.

The `get` form of the clause obtains a property value of the target or executes a `to get` handler. The `send` form specifies that a message is sent to the target.

If no target object is specified for the translated ToolBook message, the ToolBook message is sent to the current page. If a translated message is being sent to a book and the book is then saved to a different name, message translation will be canceled.

HANDLING TRANSLATED MESSAGES

A separate OpenScript handler or function must be written to respond to the translated message or to return a property value.

For example:

```
-- Defines the response to the activateApp message
to handle activateApp hwnd, wmsg, wp, lplo, lphi, lpPointer
    if wp=0
        --Sends user-defined message to current page when
        --ToolBook is deactivated
        send deactivateInstance
    end
end
```

ToolBook automatically makes these parameters available for the handler of an activateApp message:

`<hWnd>`	The `16-bit handle` of the window that originally receives the Windows message.
`<wMsg>`	The decimal value for the Windows message that is translated to the ToolBook message `activateApp`, and is dependent on the Windows message number.
`<wp>`	A value that corresponds to the `wParam` parameter for the Windows message.
`<lpLo>`	A value that corresponds to the `LOWORD` of the `lParam` parameter for the Windows message.
`<lpHi>`	A value that corresponds to the `HIWORD` of the `lParam` parameter for the Windows message.
`<lpPointer>`	A value in pointer form that is a combination of the `lpLo` and `lpHi` parameters and corresponds to `LOWORD` and `HIWORD` of the `lParam` parameter for the Windows message. This value can be passed to any ToolBook pointer `<type>` functions when a pointer parameter is required.

RETURNING VALUES

This example translates Windows message `19 (WM_QUERYOPEN)` to an OpenScript user-defined message that gets the value of a user property when the Windows message has finished processing:

```
translateWindowMessage
    on 19 get openIcon of this book return int
end translateWindowMessage
```

The words return `int` in this example indicate that an `INT` data type is returned by the user-defined function `openIcon()`.

The corresponding `to get` handler must always return a value appropriate for the `<dllType>` parameter specified in the `translateWindowMessage` control structure.

This handler must also return a value appropriate for the Windows message being translated. For the message translation of message `19 (WM_QUERYOPEN)`, the to `get openIcon` handler must return a Boolean value: `0` to prevent the instance from being activated or a nonzero number to allow the instance to be activated.

....

PARAMETERS

PARAMETER	DESCRIPTION
<winHandle>	The handle of the window that will translate the message; the default is the value of windowHandle of mainWindow.
<winMsg>	The number of the Windows message to be translated. ToolBook reserves the Windows message numbers 1124 to 1224 (WM_USER+100 to WM_USER+200) for its internal messages. Do not attempt to redefine or intercept any of these messages; doing so can cause system errors and data corruption.

<tbkMsg>	The OpenScript message to be sent before or after the Windows message is received, used in one of the following forms:
	`to handle tbkMsg hWnd, dwMsg, wParam, lParam`
	`    -- Further statements here`
	`end`
	or
	`to get tbkMsg hWnd, dwMsg, wParam, lParam`
	`    -- Further statements here`
	`end`

<tbkObj>	The target for the OpenScript message. The default value is the current page.
<dllType>	The data type of the value to be returned, specified as CHAR, BYTE, INT, WORD, LONG, DWORD, or POINTER.

EXAMPLES

```
-- Translates doubleClick so that ToolBook can receive the message
-- at Author level; overrides normal doubleClick behavior at Author
-- level, such as placing the insertion point in a field
to handle initTranslation
    translateWindowMessage for clientHandle of viewer "dialog"
        on 0x0203 send properties to this book
        --0x0203 is the decimal value of the Windows
        --message WM_LBUTTONDBLCLK
    end
end
```

translateWindowMessage32

Control Structure

SYNTAX

```
translateWindowMessage [for <winHandle>]
  on <winMsg> get <tbkMsg> [of <tbkObj>] [protected] return <dllType>
  --Further statements here
end [translateWindowMessage]

translateWindowMessage [for <winHandle>]
  --Further statements here
  on <winMsg> send <tbkMsg> [to <tbkObj>] [protected]
end [translateWindowMessage]
```

DESCRIPTION

A control structure that establishes Windows-to-OpenScript message translation and specifies whether the OpenScript message is processed instead of the window's default processing of a message.

Multiple handlers can be used for a given <winMsg>. When using translateWindow-Message32, if a given message has more than one translation, then the last translation is done first, followed by the next-to-last translation, and so on.

NOTES

Use the forward to system statement in the handler called by the on statement in the translateWindowMessage32 control structure to forward a Windows message to the ToolBook system. ToolBook calls the default Windows procedure and puts the returned value into IT.

When you forward a translated message to the ToolBook system, ToolBook calls the default Windows procedure for the message at the time the message is forwarded. If a translated message is not forwarded to the system, ToolBook does not call the default Windows procedure.

The get form of the clause obtains a property value of the target or executes a to get handler. The send form specifies that a message is sent to the target. If no target object is specified for the translated ToolBook message, the ToolBook message is sent to the current page. If a translated message is being sent to a book and the book is then saved to a different name, message translation will be canceled.

PARAMETERS

PARAMETER	DESCRIPTION
<winHandle>	The 32-bit handle of the window that will translate the message; the default is the value of windowHandle of mainWindow.
<winMsg>	The number of the Windows message to be translated. ToolBook reserves the Windows message numbers 1124 to 1224 (WM_USER+100 to WM_USER+200) for its internal messages. Do not attempt to redefine or intercept any of these messages; doing so can cause system errors and data corruption.

<tbkMsg>	The OpenScript message to be sent before or after the Windows message is received, used in one of the following forms:
	`to handle tbkMsg hWnd, dwMsg, wParam, lParam`
	`    -- Further statements here`
	`end`
	or
	`to get tbkMsg hWnd, dwMsg, wParam, lParam`
	`    -- Further statements here`
	`end`
	In the handlers above, <hWnd> is a 32-bit window handle; <wParam> and <lParam> use 32-bit Windows message parameter packing.

<tbkObj>	The target for the OpenScript message. The default value is the current page.
<protected>	An optional parameter that is allowed at the end of each on statement. Translations that use the protected parameter are only untranslated by explicit reference in untranslateWindow-Message (that is, untranslateAllWindowsMessage and untranslateWindowMessage hWnd will not remove a protected translation.) A restore system call will remove protected translations
<dllType>	The data type of the value to be returned, specified as CHAR, BYTE, INT, WORD, LONG, DWORD, or POINTER32.

EXAMPLES

```
to handle buttonClick
    translateWindowMessage32 windowHandle32 of mainWindow
        on 1225 send myMessage to self protected
        on 1225 send myMessage2 to self
    end
end

to handle myMessage2 hWnd, dwMsg, wp, lp
    request "I should be the first handler to recieve the message"
    -- Note that the following line doesn't remove the
    -- protected translation
    untranslateAllWindowsMessages for hWnd
    forward
end

to handle myMessage hWnd, dwMsg, wp, lp
    request "I should recieve the message after 'myMessage2'"
    -- this will remove the protected message
    untranslateWindowMessage 1225 send myMessage to self for hWnd
    forward
end
```

transparent

Property

DESCRIPTION A property of a draw object, recordfield, field, button, combobox, group, or paint object that specifies if an object is `transparent` or not.

NOTES You can get or set this property.

Unlike hidden objects, a `transparent` object can receive keyboard and mouse messages.

`Picture Objects` do not support the `transparent` property.

VALUES `True` or `false`.

The default setting of an objects is the value of `sysTransparent`.

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES

```
-- set the transparency of the field to true to show backgroud
transparent of field "text" = true

-- draw a new rectangle and ensure it is transparent
draw rectangle from 230,500 to 900,700
transparent of IT = true
```

truncate()

Arithmetic Values

SYNTAX `truncate(<number>)`

DESCRIPTION Returns a whole number with everything to the right of the decimal point removed. No rounding is performed.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
<number>	An expression that results in a number.

EXAMPLES

```
-- without counting for leap-years, calculate an age
ask "How many days have you been alive?"
answ = IT
wholeYears= truncate(answ / 365)
request "You are" && wholeYears && "years old."
```

tryCount

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of a question object which specifies the number of attempts made to answer this question.

NOTES You can get but not set this property in OpenScript as well as in the Actions Editor.

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

EXAMPLES

```
text of field "tries" = tryCount of group "Multiple Choice"
```

tryLimit

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of a question object which specifies the number of answer attempts allowed.

NOTES You can get but not set this property in OpenScript as well as in the Actions Editor.

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

EXAMPLES

```
text of field "tries" = tryLimit of group "Multiple Choice"
```

uncheck menuItem

Menu Command/Function

SYNTAX

```
uncheck menuItem <name | alias> [in <menu reference> [in <menu
  reference>] ... ] [at <level>]
```

DESCRIPTION Removes a `checkmark` that appears to the left of a `menu item` on the menu bar of the target window.

To add a checkmark from a menu item, use the `check menuItem` command.

NOTES You can remove a `checkmark` from a `menu item` in a submenu using the in `<menu reference>` parameter.

PARAMETERS

PARAMETER	DESCRIPTION
`<name | alias>`	The name of the menu item as it appears on the menu, or the alias assigned to the menu item.
`<menu reference>`	A valid reference to an existing menu or submenu in the target window. If more than one menu shares the same reference, ToolBook acts on the first menu from the left on the menu bar.
`<level>`	`Author`, `Reader`, or `both`. If a level is not specified, the current working level is assumed.

EXAMPLES

```
uncheck menuItem "Book List" in menu "Options" at Reader
```

```
--Sets up a menu item that can be toggled
to handle soundEffects
    if not menuItemChecked("Sound Effects")
        check menuItem "Sound Effects"
        send startSoundEffects
    else
        uncheck menuItem "Sound Effects"
        send stopSoundEffects
    end
end
```

undo

Editing

DESCRIPTION Sent to the page when `Undo` is chosen from the `Edit` menu.

You can also `send` this message using the `send undo` statement. ToolBook's default response is to reverse the user's or developer's most recent action, if that action can be undone.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
send undo
```

underline

Character Formatting

DESCRIPTION Sent to the current page when `underline` is chosen from the `Text` menu.

You can also `send` this message using the `send underline` statement. ToolBook's default response is to change the type style to underline or, if it's already underline, to turn off that style.

EXAMPLES

```
select chars 15 to 20 of text of field "list"
send underline
```

ungroup

Grouping Message

DESCRIPTION Sent to the page when you choose `Ungroup` from the `Object` menu.

You can also `send` this message using the `send ungroup` statement. ToolBook's default response is to separate the objects in the selected group so you can work with the objects individually or, if there is no group selected, to do nothing.

When a group is ungrouped, its script, user properties, and name are discarded, and the list of the objects that were in the group become the value of `selection`.

NOTE As with most all system generated messages, it is important to `forward` the message either before or after you have handled it, so that the controlling system books can also have a chance to process the message. In general, forward all system generated messages unless you have a programmatic reason not to.

EXAMPLES

```
select group "carparts"
send ungroup
```

uniqueName

Property

DESCRIPTION An object property that specifies the `unique identifier` of an object.

ToolBook assigns a unique name to each object as it is created.

NOTES You can get but you cannot set this property.

VALUES For a book, the complete file name in this form:
`book <book's file name>`

For any other object, a string that contains a unique name in the form:
`<object type name> <idNumber> of <context>`

For `hotwords`, `comboboxes`, `buttons`, `fields`, `graphics`, `OLE objects`, and `groups`, `<context>` is the `uniqueName` of the page or background where the object is located.

For pages, backgrounds, and viewers, `<context>` is the book's file name.

EXAMPLES

```
-- Display's the books uniqueName which is
-- book "c:\classroom training.tbk"
request uniqueName of this book

-- Display's the help button's uniqueName which is
-- button id 4 of page id 15
request uniqueName of button "help" of this page
```

unlinkSysBook

Notification Message

DESCRIPTION Sent when a system book is unlinked from a book as the result of being removed from a book's `sysBooks` property.

This message is sent to the system book that is unlinked, which becomes the value of `target`.

To prevent a system book from responding to this message multiple times when other system books exist, be sure to handle this message in each system book and do not `forward` it. Also, check the value of `target` to ensure that a handler is handling the message intended for a particular system book.

EXAMPLES

```
to handle unlinkSysBook
    if target = self
        send closeLoggingFiles
    end
end
```

unselect

Selection Commands

SYNTAX `unselect <object>`

DESCRIPTION Cancels the selection of a specified object and removes the object's name from the value of the `selection` property.

PARAMETERS

PARAMETER	DESCRIPTION
`<object>`	The unique name of an object.

EXAMPLES

```
unselect irregularPolygon "doubleLoop"
```

uppercase()

String Functions

SYNTAX `uppercase(<expression>)`

DESCRIPTION Converts all of the characters in the expression to uppercase.

Characters that lack an uppercase equivalent in the `ANSI` character set remain unchanged. This includes characters such as numbers.

RETURNS Returns a string value containing the modified expression.

NOTES A character that is already uppercase will remain uppercase.

ACTIONS EDITOR The Actions Editor also supports this feature.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that results in a string value.

EXAMPLES

```
-- Capitalize each word in a string
to handle buttonClick
    txt = text of field "poem"
    step k from 1 to wordCount(txt)
        curWord = word k of txt
        uc = upperCase(character 1 of curWord)
        character 1 of curWord = uc
        word k of txt = curWord
    end
    text of field "poem" = txt
end
```

useChromakey

Property

DESCRIPTION A property of a paint object or bitmap resource that specifies that portions of the bitmap will be transparent.

NOTES You can get or set this property.

This property is used in conjunction with the bitmap's `keyColor` property. If the `useChromakey` property is `true`, the portions of the bitmap that match the value of the `keyColor` property become transparent.

If setting this property for a bitmap resource, you need to set it only once for the resource and it is automatically applies everywhere the resource is used in fields or buttons.

The `useChromakey` property has no effect with resources displayed in stage objects.

For paint objects, the `fillColor` property specifies the color that is masked as transparent. For bitmap resources, the `keyColor` property performs this function.

For both paint objects and resources, the `useChromakey` property must be `true` for the transparency to take effect.

VALUES `True` or `false`.

If `true`, the fill color (or key color) cancels out the same color in the image.

EXAMPLES

```
useChromakey of paintObject "tiger" = true
fillColor of paintObject "tiger" = green
```

```
useChromakey of bitmap "fish" = true
keyColor of bitmap "fish" = magenta
```

useDialogColor

Property

DESCRIPTION To use this background property, set this property to `true` if you want the background color to match the Windows color setting for Dialog boxes. Set it to `false` if you want to control the color yourself.

Since this value can change from Windows installation to Windows installation, the color of your background will change automatically to match the user's Windows color preferences.

NOTES You can get or set this property.

VALUE `True` or `false`. Defaults to `false`.

If set to `true` any color currently specified for the background color will be ignored.

EXAMPLES

```
useDialogColor of background "course" = true
```

```
useDialogColor of this background = useDialogColor of background "x"
```

userProperties

Property

DESCRIPTION A property of all ToolBook object types which lists the names of all user properties assigned to that object.

NOTES You cannot set this property.

VALUES A comma separated list of user names currently assigned to the object.

If there are no `user properties` assigned to an object, the `userProperties` property will be `null`.

EXAMPLES

```
put userProperties of selection
```

useWindowsColors

Property

DESCRIPTION A property of draw objects, graphic objects, backgrounds, and a property of viewers that specifies whether an object is drawn in Windows colors.

Typically this is a setting that is adjusted while authoring your content but not something that you would change dynamically via OpenScript.

NOTES You can get or set this property.

ToolBook draws patterns by substituting the appropriate Windows colors for the fill and stroke colors.

When text is selected, it is drawn using inverted colors instead of Windows text colors.

ToolBook uses Windows button face and button text colors for buttons with a `borderStyle` property set to none, pushbutton, shadowed, rectangle, or `roundedRectangle`.

ToolBook uses Windows text colors and Windows background colors for check boxes and radio buttons.

For label buttons, ToolBook uses Windows background colors and button text colors.

VALUES `True` or `false`.

The default value for this setting is the setting for the `sysUseWindowsColors` property (which is `false` by default).

If `false`, ToolBook draws the object using the values of the object's `fillColor` and `strokeColor` properties.

If `true`, ToolBook draws the object using Windows colors.

Setting an object's `useWindowsColors` property to `true` does not change its `fillColor` and `strokeColor` properties.

EXAMPLES

```
useWindowsColors of button "ball" = true
```

verticalDisplayRes()

TBWIN.DLL Screen Display Function

SYNTAX

```
verticalDisplayRes()
```

DESCRIPTION Returns the vertical resolution of the display device in `pixels` (Screen Height).

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT verticalDisplayRes()
end
```

PARAMETERS No parameters

EXAMPLES

```
-- returns 600 on a 800x600 display
val = verticalDisplayRes()
```

verticalDisplaySize()

TBWIN.DLL Screen Display Function

SYNTAX `verticalDisplaySize()`

DESCRIPTION Returns the vertical resolution of the display device in `millimeters` (Screen Height).

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT verticalDisplaySize()
end
```

PARAMETERS No parameters

EXAMPLES

```
val = verticalDisplaySize()
```

vertices

Property

DESCRIPTION A property of draw objects, recordfields, fields, buttons, comboboxes, stages, graphic objects, groups, viewers or system objects that specifies the locations of an object's vertices.

NOTES You can get or set this property.

If you change the `vertices` of an irregular polygon you can effectively reshape it.

VALUES The value for `lines`, `angled lines`, `polygons`, `irregular polygons`, `arcs`, `pies`, and `curves`, the value is a comma separated list of points that define the particular object's shape. Each point is specified by two numbers representing the screen coordinate.

For all other objects, the value is a list of four numbers representing the `bounds` of the object. In fact the `bounds` of the object and the `vertices` of the object, in this case, would be identical.

`Vertices` of a `viewer` are measured in `pixels`, however the `vertices` of any other object is measured in `page units`.

EXAMPLES

```
-- change a irregular polygon from a 3 sided object to 4 sided
vertices of irregularPolygon "b3" = 0,0,3195,2115,2220,3885,4110,3885
```

```
item 1 to 2 of vertices of line "b14" = 0,12
```

viewer

Object Type

DESCRIPTION The object type name for a `viewer`, which is a type of ToolBook window that displays a page or background from any book.

NOTES The ToolBook Main window is a viewer (`viewer id 0`).

A viewer has a parent object and a parent window. The parent object of a viewer is always the book. The parent window of a popup -type viewer is usually the Main window. The parent window of a child-type viewer can be the Main window or another viewer.

To create a viewer using OpenScript, use the `send viewers` statement in the Command window. You can also create a new viewer using the `new viewer` command.

TO REFER TO A VIEWER	USE THIS SYNTAX
Currently displayed as the active window	`focusWindow`
As the target window	`targetWindow or this window`
In the current book	`viewer "widgetPalette"` `viewer ID 6`
In another book	`viewer "trees" of book "landscape.tbk"`

You can use viewers to display pages from multiple books without having to open multiple instances of ToolBook.

The `targetWindow` system property specifies the viewer in which ToolBook executes commands and searches for objects. The value for `targetWindow` is typically the same as the value for `focusWindow`, which indicates the viewer that currently has the focus. ToolBook sets the value of `targetWindow` each time the focus changes to another viewer.

Because viewers do not own other types of ToolBook objects, they are not included in, and do not change, the object hierarchy of the page or objects they are displaying.

Viewers do receive messages that specifically pertain to viewers. If a viewer does not handle a message, the viewer automatically forwards the message to its parent, which is always the book that owns the viewer, and the message continues up the hierarchy.

These built-in commands can directly affect viewers: `activate`, `close`, `hide`, `move`, new `viewer`, `open`, and `show`. The `show` and `hide` commands control whether ToolBook displays a viewer, while the `open` and `close` commands create and destroy the viewer instance.

visible

Property

DESCRIPTION

A property of draw objects, graphic objects, groups, viewers, or ToolBook system objects (the Command window, palettes, status bar, status controls, status indicators, status box, or tool bar).

The `visible` property specifies whether an object is shown or hidden.

NOTES

You can get or set this property.

If you set the `visible` property of a `viewer` that is closed to `true`, ToolBook first opens the window, then shows it.

You can use the `hide` command to set the visible property of an object to `false`.

You can use the `show` command to set the visible property of an object to `true`.

If you set the `visible` property of a group to `false`, all objects in that group are hidden. All objects in the group still retain their individual `visible` property settings however.

VALUES

`True` or `false`.

The default value is `True` for any object.

ACTIONS EDITOR

The Actions Editor also supports this feature.

EXAMPLES

```
-- both of these do the same thing
visible of button "apple" = true
show button "apple"

-- ensure all objects are visible when I enter the page
visible of objects of this page = true
```

visited

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of a page that specifies if a page has been visited by a user.

NOTES You can get but not set this property in OpenScript as well as in the Actions Editor.

In OpenScript, this can also be achieved by manipulating the `ASYM_BeenHere` page user property.

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

VALUES `True` if the page has been visited before, otherwise `false`.

EXAMPLES

```
beenHereBefore = visited of page "apple"
```

while

Control Structure

SYNTAX

```
while <expression>
    <statements>
end [while]
```

DESCRIPTION Executes the statements in the body of the control structure until the expression evaluates to `false`.

NOTES ToolBook checks the value of the expression before executing any of the statements. Consequently, the statements in the body of the control structure are not executed if the expression evaluates to `false` when ToolBook first encounters the `while` statement.

Use the `break` command to break out of a `while` control structure before it finishes executing.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that evaluates to `true` or `false`.
`<statements>`	One or more OpenScript statements.

EXAMPLES

```
pgs = pages of this book
while pgs <> null
    pop pgs into curObj
    if "printing" is in name of curObj
        skipNavigation of curObj = true
    end
end
```

white

Color Constants

DESCRIPTION Represents the color white. This is equivalent to the `RGB` value of `255,255,255` and the `HLS` value of `0,100,0`.

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES

```
rgbFill of field "list" = white
```

```
if rgbStroke of rectangle "drop area" = white
    rgbStroke of rectangle "drop area" = 0,191,0
end
```

```
strokeColor of ellipse id 14 = white
```

width

TB89ACTR.SBK Actions Editor Object Property

DESCRIPTION An Actions Editor provided property of almost all ToolBook object types that specifies the current width of that object.

NOTES You can get or set this property in OpenScript.

You can get but not set the property in the Actions Editor.

This function is provided by the TB89ACTR.SBK. To successfully use this function in Native Runtime, you must first add the TB89ACTR.SBK to your bound system books. Use the Bound System Books option in the File menu, and ensure the this TB89ACTR.SBK book is in the list of bound system books. Note that it is very possible that when you look, the TB89ACTR.SBK will already be there.

VALUES A whole numbers that specifies the `width` in `page units`.

EXAMPLES

```
width of button "foo" = 500
```

```
x = width of button "foo"
```

windowHandle

Property

DESCRIPTION A property of a viewer that specifies the `16-bit` window handle of the viewer's frame window.

NOTES This property cannot be set. All viewers contain both a frame window and a client window. The client window handle is specified by the `clientHandle` property.

VALUES A `16-bit` number assigned by Windows, or `0` if no instance of the viewer exists.

EXAMPLES

```
-- Overrides the default assignment of the parent window
to handle openWindow
    my parentHandle = windowHandle of viewer "help"
end
```

windowHandle32

Property

DESCRIPTION A property of a viewer that specifies the `32-bit` window handle of the viewer's frame window.

NOTES This property cannot be set. All viewers contain both a frame window and a client window. The client window handle is specified by the `clientHandle32` property.

VALUES A `32-bit` number assigned by Windows, or `0` if no instance of the viewer exists.

EXAMPLES

```
-- Overrides the default assignment of the parent window
to handle openWindow
    my parentHandle = windowHandle32 of viewer "help"
end
```

windows

Property

DESCRIPTION A book property that specifies a list of viewers owned by a book.

NOTES This is a read only property and cannot be set.

VALUES A comma separated list of viewers.

The `mainWindow` (`viewer id 0`) will always be included in this list.

EXAMPLES

```
-- how many viewers are defined in this book?
ct = itemCount(windows of this book)
```

with

Articles and Prepositions

DESCRIPTION Used with the following OpenScript terms for various reasons: `ask`, `draw`, `replace` `resource`, and `request`

EXAMPLES

```
ask "What color is the moon?" with "type your answer here..."
request "Do you really want to exit" with "yes" or "no"
replace resource bitmap "redApple" with "c:\grannysmith.bmp"
```

word, words

String Operators

SYNTAX

```
word <number> of <stringRef>
words <number> to <number> of <stringRef>
```

DESCRIPTION Used to refer to specific words in a text string. A `word` is defined as a sequence of characters surrounded by non-printable characters. Typically a space is the separator.

The first and last word in a text string are the exceptions to this rule, as obviously the first word will unlikely have a space preceding it, as is the last word in a string unlikely to have a space following it.

NOTES Various terms can be used with `word` including `first`, `last`, `middle`.

PARAMETERS

PARAMETER	DESCRIPTION
`<number>`	An expression that results in a number.
`<stringRef>`	An expression that results in a string value.

EXAMPLES

```
x = words 1 to 5 of text of field "list"

word 2 of field "salutation" = firstName
```

wordCount()

String Functions

SYNTAX

```
wordCount(<expression>)
```

DESCRIPTION Counts the number of words in an expression.

RETURNS Returns the number of words in a string. A `word` is defined as a sequence of characters surrounded by non-printable characters. Typically a space is the separator.

The first and last word in a text string are the exceptions to this rule, as obviously the first word will unlikely have a space preceding it, as is the last word in a string unlikely to have a space following it.

NOTES You can also count characters with `charCount()` and textlines [textline essentially means paragraph] with `textlineCount()`.

PARAMETERS

PARAMETER	DESCRIPTION
`<expression>`	An expression that results in a string value.

EXAMPLES

```
-- Capitalize each word in a string
to handle buttonClick
    txt = text of field "poem"
    step k from 1 to wordCount(txt)
        curWord = word k of txt
        uc = upperCase(character 1 of curWord)
        character 1 of curWord = uc
        word k of txt = curWord
    end
    text of field "poem" = txt
end
```

wordWrapUnits

Property

DESCRIPTION A book property that specifies the standard display units used by ToolBook for word wrapping and vertical line spacing of text.

Vertical line spacing and word wrapping can vary depending on the characteristics of the current display device.

NOTES Use the `wordWrapUnits` property to ensure that ToolBook wraps words and spaces textlines the same throughout a book, regardless of the current display device.

This capability is particularly useful in applications where you need to carefully align graphic objects with text in fields and recordfields.

VALUES A list containing a pair of numbers that specify the horizontal and vertical `page units` per `pixel`.

The value for a standard `VGA` screen is `15,15`.

The default is `0,0`. This setting specifies that ToolBook wraps words and spaces text optimally for the current display device.

If the value is anything other than `0,0`, ToolBook wraps words and spaces lines of text the same way for all display devices.

EXAMPLES

```
-- Sets the current display as the standard display to text
-- wrapping and vertical textline spacing
wordWrapUnits of this book = sysPageUnitsPerPixel
```

writeFile

File Manipulation Command

SYNTAX

```
writeFile <text> to <file name>
```

DESCRIPTION Writes text to an open file.

NOTES The file must have been explicitly created with the `createFile` command or opened with the `openFile` command.

The file remains open until a `closeFile` command is executed or the file is closed by other means.

The `writeFile` command appends text only to the end of a file, unless the `seekFile` command is used to reset the file pointer.

PARAMETERS

PARAMETER	DESCRIPTION
`<file name>`	A file name, including the path if necessary.
`<text>`	A string or a container that evaluates to a string.

EXAMPLES

```
writeFile text of field "comments" to fName

writeFile allData to fName

writeFile getAllDatat() to "logfile.log"
```

xPixelsFromUnits()

TBWIN.DLL Screen Display Function

SYNTAX `xPixelsFromUnits(<xdimension>)`

DESCRIPTION Converts a horizontal dimension between `pixels` and ToolBook `page units`.

This function is useful because the number of `page units` per `pixel` can vary with the display device.

RETURNS Returns a number of `pixels`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT xPixelsFromUnits(INT)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<xdimension>`	A whole number in the range `-32768` to `32767`.

xUnitsFromPixels()

TBWIN.DLL Screen Display Function

SYNTAX `xUnitsFromPixels(<xdimension>)`

DESCRIPTION Converts a horizontal dimension in ToolBook `page units` into a dimension in `pixels`.

This function is useful because the number of `page units` per `pixel` can vary with the display device.

RETURNS Returns a number of ToolBook `page units`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT xUnitsFromPixels(INT)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<xdimension>`	A whole number in the range `-32768` to `32767`.

yellow

Color Constants

DESCRIPTION Represents the color yellow. This is equivalent to the `RGB` value of `255,255,0` and the `HLS` value of `60,50,100`.

ACTIONS EDITOR The Actions Editor also supports this feature.

EXAMPLES

```
rgbFill of field "list" = yellow
```

```
if rgbStroke of rectangle "drop area" = yellow
    rgbStroke of rectangle "drop area" = 0,191,0
end
```

```
strokeColor of ellipse id 14 = yellow
```

yPixelsFromUnits()

TBWIN.DLL Screen Display Function

SYNTAX `yPixelsFromUnits(<ydimension>)`

DESCRIPTION Converts a vertical dimension between `pixels` and ToolBook `page units`.

This function is useful because the number of `page units` per `pixel` can vary with the display device.

RETURNS Returns a number of `pixels`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT yPixelsFromUnits(INT)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<ydimension>`	A whole number in the range `-32768` to `32767`.

yUnitsFromPixels()

TBWIN.DLL Screen Display Function

SYNTAX `yUnitsFromPixels(<ydimension>)`

DESCRIPTION Converts a vertical dimension in ToolBook `page units` into a dimension in `pixels`.

This function is useful because the number of `page units` per `pixel` can vary with the display device.

RETURNS Returns a number of ToolBook `page units`.

NOTES As of ToolBook 8.5 this is an internal function which means you can simply call the function whenever you need it. However if you are using ToolBook 8.1 or lower, you will need to link in this function at least once before calling the function. Consult LINKDLL for further explanation. You can link in the function using the following code:

```
linkdll "tbwin.dll"
    INT yUnitsFromPixels(INT)
end
```

PARAMETERS

PARAMETER	DESCRIPTION
`<ydimension>`	A whole number in the range `-32768` to `32767`.

YOUR NOTES

INDEX

B

C

D

E

INDEX

F

G

H

INDEX

I

J

K

L

M

N

O

P

Q

R

S

INDEX

T

INDEX

U

V

W

X

Y

Z